Journal of Computational Physics 545 (2026) 114447

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

An r-adaptive finite element method using neural networks for parametric self-adjoint elliptic problems

Danilo Aballay[a], Federico Fuentes[b], Vicente Iligaray[a], Ángel J. Omella[c,1], David Pardo[d,e,f], Manuel A. Sánchez[b], Ignacio Tapia[a], Carlos Uriarte[e,*]

[a] Faculty of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile
[b] Institute for Mathematical and Computational Engineering (IMC), Pontificia Universidad Católica de Chile, Santiago, Chile
[c] Departamento de Matemática Aplicada, Universidad de Zaragoza, Zaragoza, Spain
[d] University of the Basque Country (UPV/EHU), Leioa, Spain
[e] Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
[f] Basque Foundation for Science (Ikerbasque), Bilbao, Spain

A R T I C L E I N F O

Keywords:
Partial differential equations
Parametric partial differential equations
Finite element method
R-adaptivity
Ritz minimization
Neural networks

A B S T R A C T

This work proposes an r-adaptive finite element method (FEM) using neural networks (NNs). The method employs the Ritz energy functional as the loss function, currently limiting its applicability to symmetric and coercive problems, such as those arising from self-adjoint elliptic problems. The objective of the NN optimization is to determine the mesh node locations. For simplicity, these locations are assumed to form a tensor product structure in higher dimensions. The method is designed to solve parametric partial differential equations (PDEs). The resulting parametric r-adapted mesh generated by the NN is solved for each PDE parameter instance with a standard FEM. Consequently, the proposed approach retains the robustness and reliability guarantees of the FEM for each parameter instance, while the NN optimization adaptively adjusts the mesh node locations. The construction of FEM matrices and load vectors is implemented such that their derivatives with respect to mesh node locations, required for NN training, can be efficiently computed using automatic differentiation. However, the linear equation solver does not need to be differentiable, enabling the use of efficient, readily available 'out-of-the-box' solvers. The method's performance is demonstrated on parametric one- and two-dimensional Poisson problems.

1. Introduction

The finite element method (FEM) is one of the most popular techniques for solving partial differential equations (PDEs). Its strength lies in its rigorous mathematical foundation, which provides a framework for error estimation and convergence analysis. FEM solvers are known for their robustness and reliability [1–4]. A traditional FEM discretizes the spatial domain into a mesh of elements and constructs an approximated PDE solution using possibly high-order [5,6] piecewise-polynomial functions defined over this mesh. While h-adaptivity (refining the mesh size) and p-adaptivity (increasing the polynomial order) are well-established in the

* Corresponding author.
E-mail addresses: danilo.aballay@uc.cl (D. Aballay), federico.fuentes@uc.cl (F. Fuentes), vicente.iligaray@uc.cl (V. Iligaray), aomella@unizar.es (Á.J. Omella), david.pardo@ehu.eus, dpardo@bcamath.org (D. Pardo), manuel.sanchez@uc.cl (M.A. Sánchez), iptapia@uc.cl (I. Tapia), curiarte@bcamath.org (C. Uriarte).
[1] Formerly at the University of the Basque Country (UPV/EHU), where the majority of this work was conducted.

https://doi.org/10.1016/j.jcp.2025.114447
Received 26 June 2025; Received in revised form 4 October 2025; Accepted 9 October 2025
Available online 16 October 2025

FEM [7–11], r-adaptivity, which involves relocating mesh nodes while preserving the dimensionality and polynomial representability, is a less developed area [12,13]. In addition, adaptive FEMs suffer from limitations that become particularly acute in the context of parametric PDEs, especially when singular or highly localized solutions arise naturally. In these cases, generating optimal meshes for a family of problems can be a computationally intensive and time-consuming process.

In recent years, neural networks (NNs) have emerged as useful complementary tools for solving PDEs [14,15]. Methods like (Variational) Physics-Informed Neural Networks ((V)PINNs) [16–20] and Deep Ritz [21–23] take advantage of the powerful function approximation capabilities of NNs to represent the PDE solutions (see, e.g., [24]).

However, NN-based solvers also exhibit important limitations. The training process typically involves solving a highly non-convex optimization problem. Standard optimization algorithms, such as stochastic gradient descent, are prone to getting trapped in local minima, resulting in inaccurate or suboptimal solutions. This lack of guaranteed convergence is a major concern. Furthermore, unlike FEM, there is often a scarcity of rigorous theoretical guarantees regarding the convergence and accuracy of NN-based solvers [25–27]. Another significant challenge arises from the need for accurate numerical integration techniques in both (V)PINN and Deep Ritz methods. Whether evaluating the residual at collocation points or computing the energy functional, accurate numerical integration is essential [28,29]. This can be computationally demanding, especially in high dimensions, and the choice of integration scheme profoundly impacts the accuracy of the obtained solution and the stability of the training [30–32].

This work combines traditional FEM and NN-based solvers to create a more robust and efficient method for solving parametric PDEs. The core idea is to use an NN to perform r-adaptivity on the mesh of the FEM used to discretize and solve the problem. This strategy effectively addresses the key limitations of standalone NN or FEM approaches. By employing the well-posed FEM formulation, accurate integration routines, and an efficient solver for the resulting systems of linear equations, we avoid the problem of numerical integration and local minima that hinder the training of NN-based solvers. In particular, our previous work [33] proposed the use of an NN for r-adaptivity via the Ritz energy minimization; however, this work employed an NN to approximate the solution itself, which led to convergence difficulties that hindered the recovery of optimal convergence rates. In contrast, the method proposed here employs the NN solely to adapt the mesh, while the PDE is solved using a classical FEM formulation. This strategy ensures that each iteration yields the global minimum of a discretized quadratic problem that approximates the original PDE solution (see, e.g., [20,34]), providing both robustness and accuracy. Furthermore, we extend this framework to parametric PDEs by introducing an NN that captures the parametric dependence of the solution on the r-adapted mesh. Recent related works propose constructing finite-element-type solutions derived from interpolating NNs [35–37], while [38,39] propose FEM parametric solvers using NNs.

Herein, our primary contribution is the development of an r-adaptive FEM solver for parametric PDEs. The NN represents a parameter-dependent finite-element mesh, where the dimensionality in terms of the PDE coefficients may be large. This dynamic r-adaptation concentrates the mesh density on regions where the solution exhibits significant variations, such as sharp gradients, boundary layers, or singularities, leading to substantial improvements in accuracy without requiring a globally fine mesh.

In this work, we consider the Ritz minimization formulation to produce optimal r-adapted meshes for parametric PDEs. In particular, this restricts our current formulation to symmetric and coercive problems, such as those arising from self-adjoint elliptic problems. Extending our approach to non-symmetric or indefinite problems would likely require handling fine-coarse mesh settings, double-loop strategies (see, e.g., [23]), or first-order systems of least squares formulations (see, e.g., [40]), which we leave as future work. Moreover, we restrict our implementation to the lowest-order polynomials ($p = 1$) in one- and two-dimensional spatial problems and tensor-product meshes. Extending this implementation to higher-order polynomials ($p > 1$) is straightforward, while considering irregular geometries requires a more involved mesh-generation strategy that enables automatic differentiation (AD).

Our new FEM code, compatible with AD, was implemented using the JAX library [41]. Even though JAX-FEM [42] is an existing JAX-based FEM framework, it is designed for fixed meshes and thus unsuitable for r-adaptivity. Meanwhile, JAX-SSO [43], also based on FEM, is specialized to structural optimization problems involving shells. In contrast, our implementation treats the mesh node locations—central to the r-adaptive strategy—as differentiable parameters. While the assembly process requires AD, we show that the FEM linear solver does not, allowing the use of standard FEM direct or iterative solvers for improved flexibility and efficiency.

The remainder of the work is as follows. Section 2 describes the Ritz minimization procedure, the considered FEM, and the proposed r-adaptive method using NNs. Section 3 discusses our implementation and highlights its key advantages and limitations. Section 4 presents multiple numerical results that demonstrate the effectiveness and accuracy of our method on several benchmark Poisson problems in 1D and 2D. Finally, Section 5 concludes the work and outlines future lines of research.

2. Mathematical framework

We first describe the non-parametric case (Section 2.1) and then extend it to the parametric case (Section 2.2).

2.1. Non-parametric case

Let us consider the following abstract variational problem: seek $u \in \mathbb{V}$ such that

$$b(u, v) = \ell(v), \qquad \forall v \in \mathbb{V}, \tag{1}$$

where $\mathbb{V}$ is a real Hilbert space equipped with the norm $\|\cdot\|_{\mathbb{V}}$, $b : \mathbb{V} \times \mathbb{V} \longrightarrow \mathbb{R}$ is a symmetric, bounded, and coercive bilinear form, i.e., there exist boundedness and coercivity constants $M \geq \alpha > 0$ such that $M\|v\|_{\mathbb{V}}^2 \geq b(v, v) \geq \alpha\|v\|_{\mathbb{V}}^2$ for all $v \in \mathbb{V}$, and $\ell : \mathbb{V} \longrightarrow \mathbb{R}$ is a linear and continuous functional. By the Lax-Milgram theorem, (1) admits a unique solution $u \in \mathbb{V}$ satisfying $\|u\|_{\mathbb{V}} \leq \frac{1}{\alpha}\|\ell\|_{\mathbb{V}'}$, where $\|\cdot\|_{\mathbb{V}'}$ denotes the norm of the dual space $\mathbb{V}'$.

Remark 1. PDEs associated with self-adjoint elliptic differential operators can often be manipulated and formulated weakly in the form of (1), where the underlying bilinear form b will be symmetric (due to the operator being self-adjoint). Linear strictly elliptic problems of this nature will be coercive too [44]. Indeed, the coercivity constant α is also often called the ellipticity constant of the underlying operator. Examples of such problems are Poisson problems being characterized by the Laplace operator, which is self-adjoint and strictly elliptic under certain boundary conditions, i.e., there will be a coercivity constant $\alpha > 0$ related to the Poincaré constant.

Remark 2. We note that ℓ in the formulation of a PDE in the form (1) contains all the 'problem data', meaning any forcing terms in the 'right-hand side' of the PDE and any boundary conditions (either Dirichlet, Neumann, or Robin boundary conditions). Thus, the estimate $\|u\|_{\mathbb{V}} \leq \frac{1}{\alpha}\|\ell\|_{\mathbb{V}'}$ translates to a stability estimate reflecting the continuous dependence of the solution on the problem data.

2.1.1. *Ritz minimization*

Given b and ℓ from (1), define the quadratic *Ritz energy*, $\mathcal{J} : \mathbb{V} \longrightarrow \mathbb{R}$, as

$$\mathcal{J}(v) := \frac{1}{2}b(v,v) - \ell(v), \qquad v \in \mathbb{V}. \tag{2}$$

If $u \in \mathbb{V}$ is the unique solution to (1), then $b(u,v) = \ell(v)$ for any $v \in \mathbb{V}$ and, in particular, $b(u,u) = \ell(u)$. Thus, by the bilinearity and symmetry of b, for *any* $v \in \mathbb{V}$,

$$\begin{aligned}\mathcal{J}(v) - \mathcal{J}(u) &= \left(\tfrac{1}{2}b(v,v) - \ell(v)\right) - \left(\tfrac{1}{2}b(u,u) - \ell(u)\right) = \tfrac{1}{2}b(v,v) - b(u,v) - \tfrac{1}{2}b(u,u) + b(u,u)\\ &= \tfrac{1}{2}b(v,v) - \tfrac{1}{2}b(u,v) - \tfrac{1}{2}b(v,u) + \tfrac{1}{2}b(u,u) = \tfrac{1}{2}b(v-u,v) - \tfrac{1}{2}b(v-u,u)\\ &= \tfrac{1}{2}b(v-u,v-u) = \tfrac{1}{2}\|v-u\|_b^2 \geq 0\,,\end{aligned} \tag{3}$$

where $\|\cdot\|_b^2 := b(\cdot,\cdot)$ is the norm in $\mathbb{V}$ induced by the bilinear form (norm-equivalent to $\|\cdot\|_{\mathbb{V}}$ due to the coercivity and boundedness of b). Thus, $\mathcal{J}(v) > \mathcal{J}(u)$ for any $v \neq u$, meaning the unique solution $u \in \mathbb{V}$ of (1) is also the unique minimizer of $\mathcal{J}$ over $\mathbb{V}$, i.e., $u = \arg\min_{v\in\mathbb{V}} \mathcal{J}(v)$.

2.1.2. *Discretization and r-adaptivity*

For a 1D spatial domain $\Omega = (a,b)$, we consider a set of trainable variables $\theta = (\theta_1, \theta_2, \dots, \theta_n) \in \mathbb{R}^n$ and apply the *softmax* activation function to obtain a partition of unity $\delta(\theta) = (\delta_1, \delta_2, \dots, \delta_n)$ as:

$$\delta_i = \frac{\exp(\theta_i)}{\sum_{j=1}^n \exp(\theta_j)} \in (0,1), \qquad 1 \leq i \leq n, \tag{4a}$$

$$1 = \sum_{i=1}^n \delta_i. \tag{4b}$$

This partition serves to define the nodal points in $\bar{\Omega}$ as follows:

$$x_0 := a, \tag{4c}$$

$$x_i := x_{i-1} + (b-a)\delta_i, \qquad 1 \leq i \leq n. \tag{4d}$$

Note that, by construction, $a = x_0 < x_1 < x_2 < \ldots < x_n = b$.

In addition to the above parameterized nodal points for adaptivity, we may consider some additional fixed (non-adaptive) interior nodes $a < x_{n+1} < x_{n+2} < \ldots < x_N < b$ (used, for example, to define material interfaces). Hence, the final sorted list of adaptive and non-adaptive nodal points defines our FEM mesh with N elements,

$$\mathbf{x}_\theta := (\tilde{x}_0, \dots, \tilde{x}_N) = \texttt{sort}\Big[x_0, \underbrace{x_1, x_2, \dots, x_{n-1}}_{\text{adaptive nodes}}, x_n, \underbrace{x_{n+1}, x_{n+2}, \dots, x_N}_{\text{fixed nodes}}\Big]. \tag{4e}$$

Here, `sort` is the function that reorders the nodes into a consistent ascending list $x_0 = \tilde{x}_0 < \tilde{x}_1 < \ldots < \tilde{x}_N = x_n$. If an adaptive node coincides with a fixed one, duplication is removed.

We consider piecewise-linear FEM 'hat functions' $\mathbf{v}_\theta = \{v_0, \dots, v_N\}$ associated with $\mathbf{x}_\theta$ as shown in Fig. 1, satisfying $v_i(\tilde{x}_j) = \delta_{ij}$ where δ_{ij} is a Kronecker delta. One can consider a much larger set of FEM functions (e.g., piecewise polynomials of higher order), but for simplicity we limit ourselves to the most straightforward case.

Next, from the boundary conditions, we construct a discrete FEM space as the span of the relevant subset of $\{v_0, \dots, v_N\}$, which we denote by $\mathbb{V}_\theta$ to reflect its dependence on the trainable variables $\theta = (\theta_1, \theta_2, \dots, \theta_n)$. More precisely, we denote $\Lambda_{\text{dof}} \subseteq \{0, \dots, N\}$ as the indices associated with the nodes *without* assigned Dirichlet boundary conditions, and $\Lambda_D = \{0, \dots, N\} \setminus \Lambda_{\text{dof}}$ as those indices where the corresponding nodes have assigned Dirichlet boundary conditions. Thus, $\mathbb{V}_\theta = \text{span}\big(\{v_j : j \in \Lambda_{\text{dof}}\}\big)$ and the FEM solution $u_\theta \in \mathbb{V}_\theta$ to (1) may be written as

$$u_\theta(x) = \sum_{j\in\Lambda_{\text{dof}}} c_j\, v_j(x)\,, \tag{5}$$

where $\mathbf{c}_\theta = (c_j)_{j\in\Lambda_{\text{dof}}}$ is a vector of coefficients associated with the basis $\{v_j\}_{j\in\Lambda_{\text{dof}}}$, and it is given by $\mathbf{c}_\theta = \mathbf{B}_\theta^{-1}\mathbf{l}_\theta$ with $(\mathbf{B}_\theta)_{ij} := b(v_j, v_i)$ and $(\mathbf{l}_\theta)_j := \ell(v_j)$ for $i, j \in \Lambda_{\text{dof}}$. There is some abuse of notation in that $(\mathbf{B}_\theta)_{ij}$ does not exist when i or j is in Λ_D (so $\mathbf{B}_\theta$ is *not* an

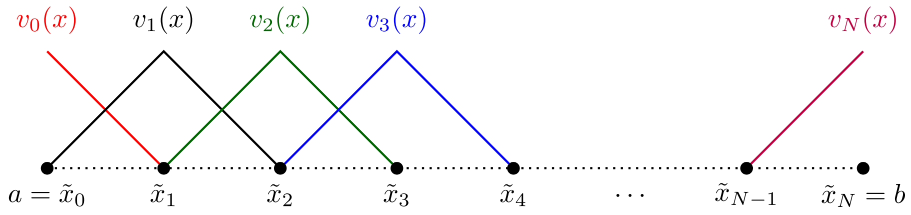


Fig. 1. Piecewise-linear FEM basis functions in $\Omega = (a, b)$.

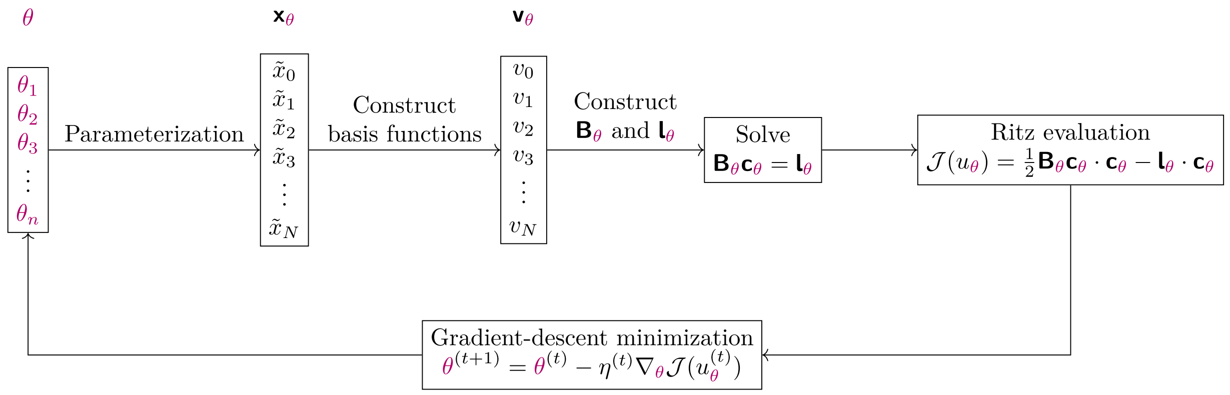

θ
θ_1 θ_2 θ_3 $\vdots$ θ_n
Parameterization
$\mathbf{x}_\theta$
$\tilde{x}_0$ $\tilde{x}_1$ $\tilde{x}_2$ $\tilde{x}_3$ $\vdots$ $\tilde{x}_N$
Construct basis functions
$\mathbf{v}_\theta$
v_0 v_1 v_2 v_3 $\vdots$ v_N
Construct $\mathbf{B}_\theta$ and $\mathbf{l}_\theta$
Solve $\mathbf{B}_\theta \mathbf{c}_\theta = \mathbf{l}_\theta$
Ritz evaluation $\mathcal{J}(u_\theta) = \frac{1}{2}\mathbf{B}_\theta \mathbf{c}_\theta \cdot \mathbf{c}_\theta - \mathbf{l}_\theta \cdot \mathbf{c}_\theta$
Gradient-descent minimization $\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} \nabla_\theta \mathcal{J}(u_\theta^{(t)})$

Fig. 2. Flowchart of the proposed r-adaptive method in the non-parametric case.

$(N+1) \times (N+1)$ matrix), but it should be clear that the $|\Lambda_{\mathrm{dof}}| \times |\Lambda_{\mathrm{dof}}|$ matrix $\mathbf{B}_\theta$ and the $|\Lambda_{\mathrm{dof}}| \times 1$ vector $\mathbf{l}_\theta$ are defined only by the indices in Λ_{dof} (which technically should require a proper relabeling to indices in $\{1, \dots, |\Lambda_{\mathrm{dof}}|\}$, but we omit these details for the sake of brevity). We also note that the Dirichlet boundary conditions enter the right-hand side $\ell(v)$ in (1), which will contain the term $-b(u_0, v)$, where $u_0(x) = \sum_{j \in \Lambda_D} c_j\, v_j(x)$ has the c_j selected to satisfy the desired Dirichlet boundary conditions, and the solution of interest (i.e., the one that solves a PDE) will then take the form $\tilde{u}(x) = u_0(x) + u_\theta(x) = \sum_{j=0}^{N} c_j v_j(x)$.

For a 2D rectangular domain $\Omega = (a^x, b^x) \times (a^y \times b^y)$, we restrict to tensor products of 1D meshes and proceed analogously, so that a set of trainable variables is given by $\theta = (\theta^x, \theta^y) \in \mathbb{R}^{n_x + n_y}$, where $\theta^x = (\theta_1^x, \theta_2^x, \dots, \theta_{n_x}^x)$ and $\theta^y = (\theta_1^y, \theta_2^y, \dots, \theta_{n_y}^y)$. Then, we apply the softmax function to obtain $\delta^x(\theta^x)$ and $\delta^y(\theta^y)$ and proceed as in (4) to obtain a set of $N_x N_y$ quadrilateral elements characterized by $(N_x+1)(N_y+1)$ 2D nodes $\mathbf{x}_\theta = \big((\tilde{x}_0, \tilde{y}_0), \dots, (\tilde{x}_{N_x}, \tilde{y}_{N_y})\big)$. With these nodes, we can then construct the associated basis of $(N_x+1)(N_y+1)$ bilinear quadrilateral hat functions $\mathbf{v}_\theta$, define the sets Λ_{dof} and Λ_D according to the boundary conditions, and proceed as described above.

Given an initial set of weights $\theta^{(0)}$, we iterate for each $t = 0, 1, \dots, T$ according to the following four-step r-adaptive procedure:

1. Given $\theta^{(t)}$, generate the mesh $\mathbf{x}_\theta^{(t)}$.
2. Construct $\mathbf{B}_\theta^{(t)}$ and $\mathbf{l}_\theta^{(t)}$.
3. Solve $\mathbf{B}_\theta^{(t)} \mathbf{c}_\theta^{(t)} = \mathbf{l}_\theta^{(t)}$.
4. Update $\theta^{(t)}$ using a gradient-descent-based algorithm applied to the Ritz loss function,

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} \nabla_\theta \mathcal{J}(u_\theta^{(t)}). \tag{6}$$

Here, $\eta^{(t)}$ denotes the iteration-dependent learning rate, which may come from sophisticated gradient-descent-based optimizers like Adam [45], and $\nabla_\theta \mathcal{J}(u_\theta^{(t)})$ is the gradient with respect to θ of the Ritz functional evaluated as

$$\mathcal{J}(u_\theta^{(t)}) = \frac{1}{2} \mathbf{B}_\theta^{(t)} \mathbf{c}_\theta^{(t)} \cdot \mathbf{c}_\theta^{(t)} - \mathbf{l}_\theta^{(t)} \cdot \mathbf{c}_\theta^{(t)}. \tag{7}$$

Fig. 2 shows a flowchart of the described r-adaptive method.

Remark 3 (No need to compute derivatives of the solver of linear equations). Viewing $\mathcal{J} = \mathcal{J}(\mathbf{B}_\theta, \mathbf{l}_\theta, \mathbf{c}_\theta)$, notice

$$\nabla_\theta \mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{B}_\theta} \frac{\partial \mathbf{B}_\theta}{\partial \theta} + \frac{\partial \mathcal{J}}{\partial \mathbf{l}_\theta} \frac{\partial \mathbf{l}_\theta}{\partial \theta} + \frac{\partial \mathcal{J}}{\partial \mathbf{c}_\theta} \frac{\partial \mathbf{c}_\theta}{\partial \theta} = \frac{1}{2} \mathbf{c}_\theta^{\mathsf{T}} \frac{\partial \mathbf{B}_\theta}{\partial \theta} \mathbf{c}_\theta - \frac{\partial \mathbf{l}_\theta}{\partial \theta} \cdot \mathbf{c}_\theta, \tag{8}$$

since $\partial \mathcal{J} / \partial \mathbf{c}_\theta = \mathbf{B}_\theta\, \mathbf{c}_\theta - \mathbf{l}_\theta = 0$. Therefore, the derivative $\partial \mathbf{c}_\theta / \partial \theta$ need not be computed, and one only needs access to $\mathbf{c}_\theta = \mathbf{B}_\theta^{-1} \mathbf{l}_\theta$, which could be solved outside any AD framework using a preferred solver of linear equations for the FEM. This decoupling allows the integration of standard, potentially non-differentiable, high-performance FEM solvers to be included within the overall optimization loop.

Remark 4. Instead of $\mathcal{J}(u_\theta) = \frac{1}{2}\mathbf{B}_\theta\mathbf{c}_\theta \cdot \mathbf{c}_\theta - \mathbf{l}_\theta \cdot \mathbf{c}_\theta$ in (7), alternative reformulations, such as $\mathcal{J}(u_\theta) = -\frac{1}{2}\mathbf{B}_\theta\mathbf{c}_\theta \cdot \mathbf{c}_\theta$ or $\mathcal{J}(u_\theta) = -\frac{1}{2}\mathbf{l}_\theta \cdot \mathbf{c}_\theta$, could be used. However, these forms are less advantageous because their partial derivative with respect to $\mathbf{c}_\theta$ generally does not vanish (i.e., $\partial\mathcal{J}/\partial\mathbf{c}_\theta \neq 0$). Consequently, Remark 3 is not applicable. Thus, whenever possible, we recommend using $\mathcal{J}(u_\theta)$ in (7) as the objective.

2.2. Parametric case

For each σ belonging to a parameter space Σ, let us consider the following variational problem: seek $u^\sigma \in \mathbb{V}$ such that

$$b^\sigma(u^\sigma, v) = \ell^\sigma(v), \qquad \forall v \in \mathbb{V}, \tag{9}$$

where $\mathbb{V}$ is a real Hilbert space equipped with the norm $\|\cdot\|_{\mathbb{V}}$, $b^\sigma : \mathbb{V}\times\mathbb{V} \longrightarrow \mathbb{R}$ is a symmetric, bounded, and coercive σ-dependent bilinear form, and $\ell^\sigma : \mathbb{V} \longrightarrow \mathbb{R}$ is a linear and continuous σ-dependent functional. Then, each σ-dependent problem in (9) admits a unique solution $u^\sigma = \arg\min_{v\in\mathbb{V}} \mathcal{J}^\sigma(v)$, where the *$\sigma$-dependent Ritz energy* is defined as

$$\mathcal{J}^\sigma(v) := \frac{1}{2} b^\sigma(v, v) - \ell^\sigma(v), \qquad v \in \mathbb{V}. \tag{10}$$

We emphasize that the range of $\mathcal{J}^\sigma$ can vary widely as a function of σ. Therefore, one might desire to balance the above so as to produce qualitatively equivalent σ-dependent functionals whose minima are close or even coincide. In a first approach, one might propose

$$\frac{\mathcal{J}^\sigma(v)}{|\mathcal{J}^\sigma(u^\sigma)|} = \frac{\frac{1}{2}b^\sigma(v,v) - \ell^\sigma(v)}{|\min_{w\in\mathbb{V}} \frac{1}{2}b^\sigma(w,w) - \ell^\sigma(w)|}, \qquad v \in \mathbb{V}, \tag{11}$$

because $\mathcal{J}^\sigma(u^\sigma)/|\mathcal{J}^\sigma(u^\sigma)| = \min_{v\in\mathbb{V}} \mathcal{J}^\sigma(v)/|\mathcal{J}^\sigma(u^\sigma)| = -1$ for any $\sigma \in \Sigma$ whenever $\mathcal{J}^\sigma(u^\sigma) \neq 0$, which occurs when $u^\sigma \neq 0$. However, this requires knowing the value of $\mathcal{J}^\sigma(u^\sigma)$ beforehand, which is often precisely what one wishes to compute with precision. With this in mind, we can instead use a proxy that is cheaply computed and that aims to achieve the same goal of balancing $\mathcal{J}^\sigma$ on different values of σ. For $N \in \mathbb{N}$ and $h = \frac{1}{N} > 0$, we propose considering the *σ-balanced Ritz energy* as

$$\tilde{\mathcal{J}}^\sigma(v) := \frac{\mathcal{J}^\sigma(v)}{|\mathcal{J}^\sigma(u_h^\sigma)|}, \qquad v \in \mathbb{V}, \tag{12}$$

where $u_h^\sigma = \arg\min_{v_h\in\mathbb{V}_h} \mathcal{J}^\sigma(v_h)$ is the 'discrete' solution to the minimization problem over a finite-dimensional subspace $\mathbb{V}_h \subseteq \mathbb{V}$ coming from a *uniform* discretization of size $h = \frac{1}{N}$ of the spatial domain Ω. As a result, we expect $|\tilde{\mathcal{J}}^\sigma(u^\sigma)| = \mathcal{O}(1)$ across different values of σ.

Remark 5. From Remark 4, we have that $\mathcal{J}^\sigma(u_h^\sigma) = -\frac{1}{2}(\mathbf{c}_h^\sigma)^\top\mathbf{B}_h^\sigma\mathbf{c}_h^\sigma = -\frac{1}{2}b^\sigma(u_h^\sigma, u_h^\sigma) < 0$ as long as $u_h^\sigma \neq 0$. Therefore, $\mathcal{J}^\sigma(u_h^\sigma)$ is 'always' negative and $\tilde{\mathcal{J}}^\sigma$ is well defined.

2.2.1. Discretization and r-adaptivity

For a 1D spatial domain $\Omega = (a, b)$, we consider the nodal-point discretization $\mathbf{x}_\theta^\sigma$ to be σ-dependent by first applying a fully-connected NN to $\sigma \in \Sigma$ and then utilizing our previous mesh generation scheme:

$$\begin{aligned}
\mathsf{z}_0 &:= \sigma \in \Sigma, && \text{(13a)}\\
\mathsf{z}_l(\sigma) &:= \varphi(\mathsf{W}_l\, \mathsf{z}_{l-1}(\sigma) + \mathsf{b}_l), \qquad 1 \le l \le L-1, && \text{(13b)}\\
\delta(\sigma) &:= (\delta_1(\sigma), \delta_2(\sigma), \dots, \delta_n(\sigma)) = \mathrm{softmax}(\mathsf{W}_L\, \mathsf{z}_L(\sigma)), && \text{(13c)}\\
x_0 &:= a, && \text{(13d)}\\
x_i(\sigma) &:= x_{i-1}(\sigma) + (b-a)\delta_i(\sigma), \qquad 1 \le i \le n, && \text{(13e)}\\
\mathbf{x}_\theta^\sigma &:= \mathtt{sort}\big[x_0, x_1(\sigma), x_2(\sigma), \dots, x_{n-1}(\sigma), x_n, x_{n+1}, \dots, x_N\big]. && \text{(13f)}
\end{aligned}$$

Here, $\theta = (\mathsf{W}_1, \mathsf{b}_1, \mathsf{W}_2, \mathsf{b}_2, \dots, \mathsf{W}_{L-1}, \mathsf{b}_{L-1}, \mathsf{W}_L)$ is the set of trainable variables (weights and biases) of the NN, L is its depth, and φ is an activation function that acts componentwise.

As a result, θ delivers σ-dependent meshes $\mathbf{x}_\theta^\sigma$, which we can subordinate to FEM bases $\mathbf{v}_\theta^\sigma = \{v_0^\sigma, \dots, v_N^\sigma\}$, as described in Section 2.1.2. The σ-dependent discrete finite dimensional spaces $\mathbb{V}_\theta^\sigma = \mathrm{span}\big(\{v_j^\sigma : j \in \Lambda_{\mathrm{dof}}^\sigma\}\big)$ are then defined after having properly labeled the nodes associated to Dirichlet boundary conditions in Λ_D^σ, leaving the remaining indices as $\Lambda_{\mathrm{dof}}^\sigma$. The optimal FEM solution to (9) in $\mathbb{V}_\theta^\sigma$ is then given by

$$u_\theta^\sigma(x) = \sum_{j\in\Lambda_{\mathrm{dof}}^\sigma} c_j^\sigma\, v_j^\sigma(x), \tag{14}$$

where $\mathbf{c}_\theta^\sigma = (c_j^\sigma)_{j\in\Lambda_{\mathrm{dof}}}$ is a vector of coefficients associated with the basis $\{v_j^\sigma\}_{j\in\Lambda_{\mathrm{dof}}^\sigma}$, and is given by $\mathbf{c}_\theta^\sigma = (\mathbf{B}_\theta^\sigma)^{-1}\mathbf{l}_\theta^\sigma$ with $(\mathbf{B}_\theta^\sigma)_{ij} := b^\sigma(v_j^\sigma, v_i^\sigma)$ and $(\mathbf{l}_\theta^\sigma)_j := \ell^\sigma(v_j^\sigma)$ for $i, j \in \Lambda_{\mathrm{dof}}^\sigma$. The generalization to 2D and 3D tensor-product meshes is straightforward (see Section 2.1.2).

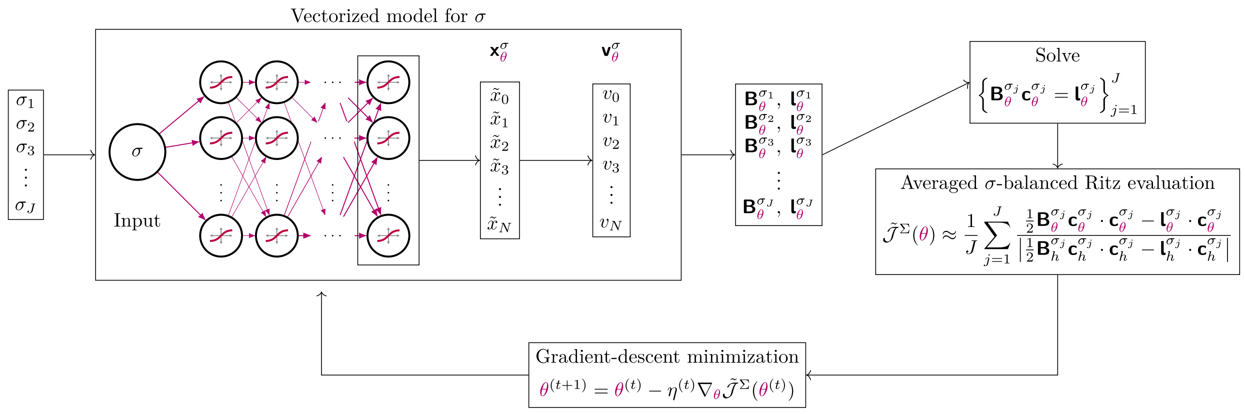

Vectorized model for σ
σ_1
σ_2
σ_3
σ_J
σ
Input
$\mathbf{x}_\theta^\sigma$
$\tilde{x}_0$ $\tilde{x}_1$ $\tilde{x}_2$ $\tilde{x}_3$ $\tilde{x}_N$
$\mathbf{v}_\theta^\sigma$
v_0 v_1 v_2 v_3 v_N
$\mathbf{B}_\theta^{\sigma_1}, \mathbf{l}_\theta^{\sigma_1}$
$\mathbf{B}_\theta^{\sigma_2}, \mathbf{l}_\theta^{\sigma_2}$
$\mathbf{B}_\theta^{\sigma_3}, \mathbf{l}_\theta^{\sigma_3}$
$\mathbf{B}_\theta^{\sigma_J}, \mathbf{l}_\theta^{\sigma_J}$
Solve
$\left\{\mathbf{B}_\theta^{\sigma_j}\mathbf{c}_\theta^{\sigma_j} = \mathbf{l}_\theta^{\sigma_j}\right\}_{j=1}^J$
Averaged σ-balanced Ritz evaluation
$\tilde{\mathcal{J}}^\Sigma(\theta) \approx \frac{1}{J}\sum_{j=1}^J \frac{\frac{1}{2}\mathbf{B}_\theta^{\sigma_j}\mathbf{c}_\theta^{\sigma_j}\cdot\mathbf{c}_\theta^{\sigma_j} - \mathbf{l}_\theta^{\sigma_j}\cdot\mathbf{c}_\theta^{\sigma_j}}{\left|\frac{1}{2}\mathbf{B}_h^{\sigma_j}\mathbf{c}_h^{\sigma_j}\cdot\mathbf{c}_h^{\sigma_j} - \mathbf{l}_h^{\sigma_j}\cdot\mathbf{c}_h^{\sigma_j}\right|}$
Gradient-descent minimization
$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)}\nabla_\theta\tilde{\mathcal{J}}^\Sigma(\theta^{(t)})$

Fig. 3. Flowchart of the proposed r-adaptive method in the parametric case.

The space $\mathbb{V}_\theta^\sigma$ is conforming to a larger infinite-dimensional space $\mathbb{V}$, so $\mathbb{V}_\theta^\sigma \subseteq \mathbb{V}$, and the solution u^σ to (9) satisfies the following optimality condition in relation to its $\mathbb{V}_\theta^\sigma$-restricted solution $u_\theta^\sigma \in \mathbb{V}_\theta^\sigma$,

$$\tilde{\mathcal{J}}^\sigma(u^\sigma) = \min_{v\in\mathbb{V}} \tilde{\mathcal{J}}^\sigma(v) \le \min_{v\in\mathbb{V}_\theta^\sigma} \tilde{\mathcal{J}}^\sigma(v) = \tilde{\mathcal{J}}^\sigma(u_\theta^\sigma), \qquad \text{for every } \sigma\in\Sigma \text{ and any } \theta. \tag{15}$$

Then, we parameterize the balanced Ritz functional across all σ as follows:

$$\theta \mapsto \tilde{\mathcal{J}}^\Sigma(\theta) := \int_\Sigma \tilde{\mathcal{J}}^\sigma(u_\theta^\sigma)\, d\mu(\sigma), \qquad \tilde{\mathcal{J}}^\sigma(\,\cdot\,) = \frac{\mathcal{J}^\sigma(\,\cdot\,)}{|\mathcal{J}^\sigma(u_h^\sigma)|}, \tag{16}$$

where μ is an appropriate measure for Σ, and $u_h^\sigma = \arg\min_{v_h\in\mathbb{V}_h} \mathcal{J}^\sigma(v_h)$ is the solution at the uniform mesh of size $h = \frac{1}{N}$ (with h selected to be compatible with fixed material interfaces, when present). By (15), it is immediate that $\tilde{\mathcal{J}}_{\mathbb{V}}^\Sigma := \int_\Sigma \tilde{\mathcal{J}}^\sigma(u^\sigma)\, d\mu(\sigma)$ is a lower bound for $\tilde{\mathcal{J}}^\Sigma(\theta)$ in (16), i.e., $\tilde{\mathcal{J}}_{\mathbb{V}}^\Sigma \le \tilde{\mathcal{J}}^\Sigma(\theta)$ for any θ.

In practice, for each iteration of the gradient-descent optimization, we approximate the above integral via the following finite weighted average:

$$\tilde{\mathcal{J}}^\Sigma(\theta) \approx \frac{1}{J}\sum_{j=1}^J \tilde{\mathcal{J}}^{\sigma_j}(u_\theta^{\sigma_j}), \tag{17}$$

where $\{\sigma_1, \sigma_2, \dots, \sigma_J\} \subset \Sigma$ is a finite sample of parameters coming from some problem-dependent distribution (in the simplest case, a uniform one).

Fig. 3 shows the flowchart for our r-adaptive method for parametric problems.

2.3. *On exact computation of integrals within r-adaptivity*

Before proceeding further, we alert the reader on a delicate issue regarding the accurate computation of the Ritz energy functional $\mathcal{J}(u)$, which translates to evaluating the forms $b(u,u)$ and $\ell(u)$, normally containing integrals. In FEM, these are usually calculated elementwise via standard quadrature rules, such as Gauss-Legendre, which are exact for typical polynomial FEM integrands, but only *approximate* non-polynomial ones which may appear in $b(u,u)$ or $\ell(u)$ under certain material heterogenities and forcings. While this quadrature error diminishes with h-refinements [4], it is not negligible for r-adaptivity, as we show next.

Consider solving $-u''(x) = f(x) := \frac{2\alpha^3(x-s)}{(1+\alpha^2(x-s)^2)^2}$ in $\Omega = (0,1)$ with $u(0) = 0$ and $u'(1) = g := \frac{\alpha}{1+\alpha^2(1-s)^2}$, whose exact solution is $u(x) = \arctan(\alpha(x-s)) + \arctan(\alpha s)$. As noted in Section 2.1.1, this solution is the unique minimizer of the Ritz functional $\mathcal{J}(v) = \frac{1}{2}b(v,v) - \ell(v)$, where $b(v,v) := \int_\Omega |\nabla v|^2\, \mathrm{d}\Omega$ and $\ell(v) := \int_\Omega f\,v\, \mathrm{d}\Omega - v(1)g$ for $v\in\mathbb{V} := \{v\in H^1(\Omega) : v(0) = 0\}$. Now, consider mesh nodes $0 = x_0 < x_1 < \dots < x_N = 1$, elements $\Omega^e = (x_{e-1}, x_e)$ for $e = 1,\dots,N$, and corresponding discrete space $\mathbb{V}_\theta = \operatorname{span}\big(\{v_1,\dots,v_N\}\big) \subseteq \mathbb{V}$ spanned by the associated hat functions v_j, as described in Section 2.1.2. For simplicity, and to illustrate the main point, let $\alpha = 50$, $s = 0.5$, and $N = 10$, and fix all nodal points x_j at a uniform mesh (i.e., $x_j = \frac{j}{10}$) except $x_5 = 0.5 + \theta$, which will then be a θ-dependent r-adaptable node that modifies $\mathbb{V}_\theta$. Thus, $\min_{v_\theta\in\mathbb{V}_\theta} \mathcal{J}(v_\theta)$ will vary as a function of θ, whose landscape is depicted in red in Fig. 4. This curve, showing two local minima in the top-right panel, lies entirely above the true global minimum $\mathcal{J}(u)$. An r-adaptive strategy optimizing θ should approach one of these local minima. However, this relies on being able to compute $\mathcal{J}(v_\theta)$ exactly.

Let $\mathcal{J}_q(v_\theta)$ be the *approximation* of $\mathcal{J}(v_\theta)$ resulting from computing the elementwise integrals in $\mathcal{J}(v_\theta)$ for $v_\theta\in\mathbb{V}_\theta$ with two-point Gauss-Legendre quadrature. Clearly, $\mathcal{J}_q(v_\theta) = \frac{1}{2}b(v_\theta,v_\theta) - \ell_q(v_\theta)$ since $b(v_\theta,v_\theta)$ is computed exactly. However, $\ell_q(v_\theta) \ne \ell(v_\theta)$ and this has a dramatic effect on the Ritz energy landscape, which is illustrated in blue in Fig. 4. This new landscape has different local minima, and, notably, those minima lie orders of magnitude below the theoretical global minimum $\mathcal{J}(u) = \min_{v\in\mathbb{V}} \mathcal{J}(v)$. Note that a small change in θ, for instance, $\theta = -0.005$, results in a huge jump in $\min_{v_\theta\in\mathbb{V}_\theta} \mathcal{J}_q(v_\theta)$, when compared to $\min_{v_\theta\in\mathbb{V}_\theta} \mathcal{J}(v_\theta)$. If a NN is

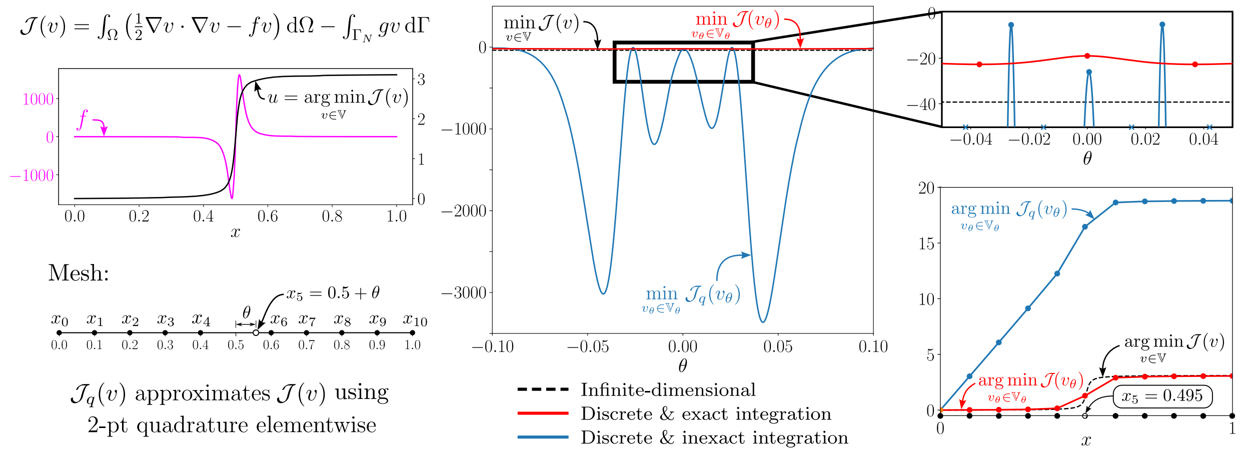


Fig. 4. For a Poisson problem $-u''(x) = f(x)$ with solution $u(x) = \arctan(\alpha(x-s)) + \arctan(\alpha s)$ for $\alpha = 50$ and $s = 0.5$, as shown in the left panel, we illustrate the landscapes of the minimum Ritz energy over piecewise-linear functions as a single mesh-node position is varied. The minimal Ritz energy landscapes corresponding to the exact and inexact integration and their associated minimizers are shown in red and blue, respectively, in the central and right panels. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

used to solve this problem, the underlying optimizer will quickly approach the minimizers in the degraded landscape, instead of the theoretical ones. Moreover, the resulting minimizer will be quite far from the theoretical optimizer, as illustrated in the bottom-right panel of Fig. 4.

Therefore, it is crucial to evaluate $\mathcal{J}(v_\theta)$ accurately. One solution is to use highly-accurate quadrature rules and hope that the resulting $\mathcal{J}_q(v_\theta)$ is sufficiently close to $\mathcal{J}(v_\theta)$. Alternatively, if possible, one may implement analytical integration routines to evaluate $\mathcal{J}(v_\theta)$ exactly, or limit oneself to problems with suitable piecewise-polynomial integrands that can be evaluated exactly with (sufficiently accurate) numerical quadrature. In this work, most non-polynomial expressions for the integrands lent themselves to analytic integration, so, when possible, we implemented specialized routines to integrate them exactly and avoid this potential issue.

3. Implementation of the JAX-based finite element code

3.1. Automatic differentiation and just-in-time compilation

The code was implemented in Python using the library JAX [41,46] to: (a) allow for automatic differentiation (AD) of its components, which is necessary in order to train the NN using stochastic gradient descent, and (b) enable XLA (Accelerated Linear Algebra) usage via just-in-time (JIT) compilation. It is available at the GitHub repository [47], which contains all the details described below and the numerical experiments of Section 4.

As mentioned in the introduction, there are other JAX-based FEM codes such as JAX-FEM [42] and JAX-SSO [43]. JAX-SSO [43] is specialized to shape and topology optimization problems involving shells. Meanwhile, JAX-FEM [42] is developed with inverse problems in mind, often involving nonlinear forward problems coming from computational mechanics, and is built to solve problems efficiently on a fixed given mesh (using JAX's AD). Our r-adaptive method, however, requires differentiating the loss function with respect to the parameters that determine the mesh node positions. Because this implies optimizing a variable mesh geometry, it fundamentally changes how FEM basis functions must be handled, making it incompatible with JAX-FEM's fixed-mesh structure. This forced us to build our implementation from the ground up.

To benefit from JAX's JIT compilation, a careful implementation is required. Indeed, control flow constructs such as `if` statements or loops can interfere with tracing when their conditions or iteration bounds depend on runtime values (e.g., JAX arrays), resulting in errors or recompilation. Thus, most `if` statements disallow exploiting the advantages of JIT compilation. To handle these situations, one must use a suite of JAX functions that are compatible with both JIT and AD. This presents some challenges, but despite this limitation, the existing suite is sufficient to achieve the desired objectives. Further implementation details follow.

3.2. Dirichlet node labeling

The mesh is composed of both fixed nodes (e.g., those needed for interfaces) and NN-trained adaptable nodes (see Section 2.1.2). After each update to the adaptable nodes, the fixed nodes are re-added and the full list is re-sorted as detailed in Section 2.1.2.

A key challenge is labeling Dirichlet boundary nodes (see Λ_D in Section 2.1.2), especially in complex cases like the L-shape domain or in the presence of interior interfaces, where node classification must be updated dynamically based on position. Standard conditional logic (namely, `if` statements) for this dynamic labeling is incompatible with JAX's JIT compiler. To address this, we employ JIT- and AD-compatible JAX-native functions from the `jax.numpy` module, like `jax.numpy.sign`, to create masks that identify nodes based on their coordinates. For example, to determine if $x \geq \alpha$ one can use $\chi_{[\alpha,\infty)}(x) = \operatorname{sgn}\big(1 - \frac{1}{2}\big(1 + \operatorname{sgn}(-(x-\alpha))\big)\big)$ as an indicator function.

3.3. Assembly, element routine, and boundary conditions

While JAX supports sparse matrix formats like Coordinate format (COO) and Compressed Sparse Row (CSR), custom routines were needed for some format conversions. In addition, implementing Dirichlet boundary conditions presented challenges due, in part, to the difficulty in isolating submatrices of the sparse stiffness matrix without using JIT-incompatible conditional statements. The solution involved modifying the element stiffness matrix directly (adding auxiliary rows and columns) during the element routine. Existing Dirichlet nodes were then carefully treated by saving the necessary information for inclusion in the force vector. This approach avoided conditional statements and dynamically-sized arrays to improve efficiency and leverage the advantages of JIT compilation, and produced a sparse stiffness matrix with the desired structure while facilitating the incorporation of the relevant Dirichlet and Neumann boundary condition data during the assembly of the force vector.

Material properties were assumed constant per element but could be heterogeneous across the domain. These were handled using JIT-compatible techniques via sign function tricks to produce indicator functions that facilitate spatial localization, as described above in Section 3.2. Element integrals of shape functions in the stiffness matrix, which only involve polynomials and piecewise constant material properties, were computed exactly using numerical quadrature. However, element integrals in the force vector sometimes involve integrands that are *not* polynomial, which, as described in Section 2.3, can lead to major issues. In these cases, for our examples, we implemented specialized routines that evaluate the integrals analytically, thus ensuring their exact evaluation. The one exception was the example in Section 4.3.1, where we used Gauss-Legendre quadrature with 50^2 integration points per element.

3.4. Solvers

We used sparse linear algebra routines suitable for the matrices arising in FEM to solve the linear systems $\mathbf{B}_\theta \mathbf{c}_\theta = \mathbf{l}_\theta$. We considered the `jax.experimental.sparse.linalg` library, noting its primary support for CUDA GPU backends at the time of implementation, with a fallback to `scipy.sparse.linalg.spsolve` for CPU execution. These sparse solver routines are compatible with JAX's AD but lack support for batched operations (e.g., `vmap`). This may deteriorate performance when considering large batches of parameter samples during the training phase. However, as noted in Remark 3, it is not necessary to automatically differentiate with respect to $\mathbf{c}_\theta$, and this can be implemented in JAX by overriding its default AD behavior via careful usage of the `@jax.custom_vjp` decorator. Thus, if large batches are being considered or the aforementioned solvers are unsuitable, one may exploit JAX's batched CSR matrix format features and replace the solver by a more efficient (possibly iterative) one to gain efficiency. In practice, this includes the use of external high-performance solvers such as PETSc [48] or MUMPS [49], which can significantly increase the computational efficiency in large-scale settings.

4. Numerical results

4.1. Benchmark problems, training procedure, and relative errors

4.1.1. Benchmark problems

We consider two elliptic boundary-value model problems, distinguished by the parameter dependence residing in either the left- or right-hand side of Poisson's equation:

$$\begin{cases} -\Delta u^\sigma = f^\sigma, & \text{in } \Omega, \\ u^\sigma = 0, & \text{on } \Gamma_D, \\ \nabla u^\sigma \cdot n = g^\sigma, & \text{on } \Gamma_N, \end{cases} \tag{18a}$$

$$\begin{cases} -\nabla \cdot \big(\sigma \nabla u^\sigma\big) = f, & \text{in } \Omega, \\ u^\sigma = 0, & \text{on } \Gamma_D, \\ \sigma \nabla u^\sigma \cdot n = g, & \text{on } \Gamma_N, \end{cases} \tag{18b}$$

where Ω is the spatial domain, Γ_D and Γ_N are the Dirichlet and Neumann boundaries, respectively, $g \in L^2(\Gamma_N)$ is the Neumann condition, and n is the outward unit normal vector on Γ_N. Regarding the parametric behavior, $f^\sigma \in L^2(\Omega)$ is σ-dependent in (18a), while $0 < \sigma \in L^\infty(\Omega)$ is piecewise constant and $f \in L^2(\Omega)$ is fixed in (18b). The variational formulations of (18) then read as: seek $u^\sigma \in H^1_D(\Omega) = \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}$ such that

$$b(u^\sigma, v) := \int_\Omega \nabla u^\sigma \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f^\sigma v \, \mathrm{d}\Omega + \int_{\Gamma_N} g^\sigma v \, \mathrm{d}\Gamma =: \ell^\sigma(v), \qquad \forall v \in H^1_D(\Omega). \tag{19a}$$

$$b^\sigma(u^\sigma, v) := \int_\Omega \sigma \nabla u^\sigma \cdot \nabla v \, \mathrm{d}\Omega = \int_\Omega f \; v \, \mathrm{d}\Omega + \int_{\Gamma_N} g \; v \, \mathrm{d}\Gamma =: \; \ell(v), \qquad \forall v \in H^1_D(\Omega). \tag{19b}$$

We highlight that the $\|\cdot\|_b$ norm is σ-independent in (18a), i.e., $\|\cdot\|_b^2 = b(\cdot,\cdot) = \big(\nabla(\cdot), \nabla(\cdot)\big)_{L^2(\Omega)} = |\cdot|^2_{H^1(\Omega)}$, while it is σ-dependent in (18b), i.e., $\|\cdot\|^2_{b^\sigma} = b^\sigma(\cdot,\cdot) = \big(\sigma\nabla(\cdot), \nabla(\cdot)\big)_{L^2(\Omega)} = \|\sqrt{\sigma}\nabla(\cdot)\|^2_{L^2(\Omega)}$.

In what follows, $\Omega \subseteq \mathbb{R}^d$, where d is the spatial dimension. We will refer to the exact solution as $u = u^\sigma$, whereas u_θ denotes the r-adaptive solution obtained via our θ-based discretization for non-parametric problems (Section 2.1.2) trained at a specific value of σ, u_θ^σ is the r-adaptive solution at σ using our NN-based discretization for parametric problems (Section 2.2.1), and $u_h = u_h^\sigma$ indicates the FEM solution on a uniform mesh of size $h = \frac{1}{N}$.

4.1.2. Neural network architecture and training procedure

For the case of parametric NNs, the parameter space Σ often comprises multiple individual parameters. Mathematically, this space is represented as a Cartesian product $\Sigma = \Sigma_1 \times \cdots \times \Sigma_P$. Consequently, any specific parameter configuration σ within this space is a tuple $\sigma = (s_1, \dots, s_P)$. When we generate a finite sample set $\{\sigma_1, \dots, \sigma_J\} \subset \Sigma$, such as the one used in (17), each sample tuple $\sigma_i = (s_1^{(i)}, \dots, s_P^{(i)})$ is formed by drawing its p-th component, $s_p^{(i)}$, from the corresponding component space Σ_p.

For our parameter data, we sampled each Σ_p and constructed a set of tuples $\sigma \in \Sigma$ as a 'grid' of all possible combinations. Each space Σ_p takes values in a compact interval, and was carefully selected, especially when the relevant parameter led to singular or localized solution behavior close to an interval extreme. In these cases, the associated distribution was often biased toward that extreme, to improve the accuracy of the trained parametric NN in that vicinity (see the examples below for the specific distributions used).

Moreover, with the same purpose, the 'corners' in the grid of parameter tuples, i.e., $\sigma = (s_1, \dots, s_P)$, where each s_p is an interval extreme, were explicitly selected as part of the training set $\{\sigma_1, \sigma_2, \dots, \sigma_J\} \subset \Sigma$. The training set, which contained 70 % of the grid tuples, was otherwise randomly selected, leaving behind 30 % of grid tuples as the test set.

We used the Adam optimizer [45] for training with learning rates (LR) in the range of 10^{-5} to 10^{-2}, and trained the parametric NNs carefully until we observed convergence, which was monitored using a fixed test subset of ten randomly chosen tuples from the test set. This sometimes required an epoch-dependent LR-strategy, where an epoch is a sweep over the entire training set. The specific LR-strategy is detailed in each example below.

Regarding the NN architecture used for the parametric NN examples, the input layer size is P, which corresponds to the σ-tuple size, and it is followed by two densely connected layers with 10 nodes each, and a final output layer of the size of the number of r-adaptable nodes (see the size of $\delta(\sigma)$ in (13c)). We used LeCun initialization [50], and hyperbolic tangent as activation functions (i.e., φ in (13b)) at every neuron. Lastly, for the non-parametric NN described in Section 2.1.2, initial weights were selected as $\theta^{(0)} = (0, \dots, 0)$.

4.1.3. Relative errors

For the non-parametric examples, we report the relative errors using the $\|\cdot\|_b$ norm. Note that, in view of (3), $\mathcal{J}(u_\theta) - \mathcal{J}(u) = \frac{1}{2}\|u_\theta - u\|_b^2$, and that $\ell(u) = b(u,u) = \|u\|_b^2$, so $\mathcal{J}(u) = \frac{1}{2}b(u,u) - \ell(u) = -\frac{1}{2}\|u\|_b^2$. Thus, we define the relative errors for the r-adaptive mesh and the corresponding uniform grid as

$$e_\theta := \left(\frac{\mathcal{J}(u) - \mathcal{J}(u_\theta)}{\mathcal{J}(u)}\right)^{\frac{1}{2}} = \frac{\|u - u_\theta\|_b}{\|u\|_b}, \tag{20a}$$

$$e_h := \left(\frac{\mathcal{J}(u) - \mathcal{J}(u_h)}{\mathcal{J}(u)}\right)^{\frac{1}{2}} = \frac{\|u - u_h\|_b}{\|u\|_b}. \tag{20b}$$

For the parametric examples, we proceed analogously, noting that on this occasion we can use either the σ-balanced or usual Ritz energy for the σ-specific relative error computation,

$$e_\theta^\sigma := \left(\frac{\tilde{\mathcal{J}}^\sigma(u^\sigma) - \tilde{\mathcal{J}}^\sigma(u_\theta^\sigma)}{\tilde{\mathcal{J}}^\sigma(u^\sigma)}\right)^{\frac{1}{2}} = \left(\frac{\mathcal{J}^\sigma(u^\sigma) - \mathcal{J}^\sigma(u_\theta^\sigma)}{\mathcal{J}^\sigma(u^\sigma)}\right)^{\frac{1}{2}} = \frac{\|u^\sigma - u_\theta^\sigma\|_{b^\sigma}}{\|u^\sigma\|_{b^\sigma}}, \tag{21a}$$

$$e_h^\sigma := \left(\frac{\tilde{\mathcal{J}}^\sigma(u^\sigma) - \tilde{\mathcal{J}}^\sigma(u_h^\sigma)}{\tilde{\mathcal{J}}^\sigma(u^\sigma)}\right)^{\frac{1}{2}} = \left(\frac{\mathcal{J}^\sigma(u^\sigma) - \mathcal{J}^\sigma(u_h^\sigma)}{\mathcal{J}^\sigma(u^\sigma)}\right)^{\frac{1}{2}} = \frac{\|u^\sigma - u_h^\sigma\|_{b^\sigma}}{\|u^\sigma\|_{b^\sigma}}. \tag{21b}$$

The mean and maximum of these errors are then computed over the σ's in the training and test datasets. Lastly, as the optimizer iterated, we used e_θ to monitor the convergence of the non-parametric case described in Section 2.1.2, whereas for the parametric case we used the average of e_θ^σ over the ten σ-tuples in a fixed test subset $\{\sigma_t\}_{t=1}^{10}$,

$$e_\theta^{\text{test}} := \frac{1}{10}\sum_{t=1}^{10} e_\theta^{\sigma_t}. \tag{22}$$

We emphasize that exact integration is critical for the above equalities to hold, as they involve Ritz energy computations, especially when the load term ℓ is challenging to integrate exactly (recall Section 2.3).

4.2. One-dimensional experiments

We present three one-dimensional benchmark problems, in both their non-parametric and parametric versions. In all examples, the domain is $\Omega = (0,1)$. In light of the comments in Section 2.3, all integrals were computed exactly, in some cases via analytic integration

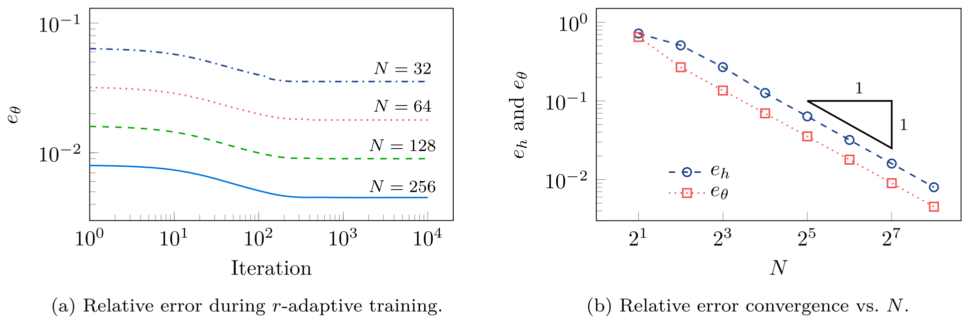


Fig. 5. Training histories during r-adaptive optimization for various mesh sizes N (left panel) and relative errors of the uniform and r-adapted finite element solutions as a function of N (right panel) for the non-parametric model problem (18a) with manufactured solution $u(x) = \arctan(10(x-0.5)) + \arctan(5)$.

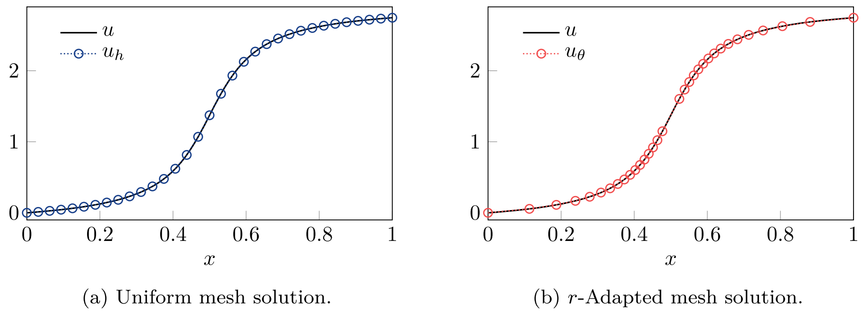


Fig. 6. Finite element solutions using a uniform and r-adapted mesh according to the results in Fig. 5 for $N = 32$.

routines specialized to the given problem. We utilized the Adam optimizer in all of our experiments, both in the non-parametric and parametric cases. The linear system is solved using `jax.experimental.sparse.linalg` within the JAX AD framework in the non-parametric examples, while we avoided differentiating with respect to the solution and used `scipy.sparse.linalg.spsolve` in the parametric ones, as detailed in Section 3.4, using batches of size 10.

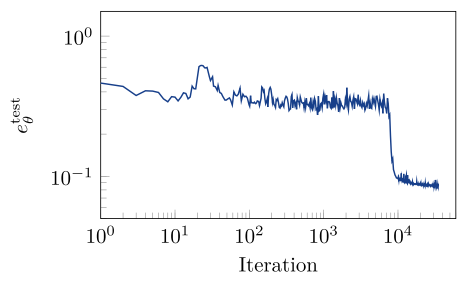
4.2.1. Arctangent sigmoid functions

We begin by considering the model problem (18a), using an arctangent sigmoid function as a manufactured solution that features a sharp internal gradient given by $u(x) = \arctan(\alpha(x-s)) + \arctan(\alpha s)$. The associated forcing function is $f(x) = -u''(x) = \frac{2\alpha^3(x-s)}{(1+\alpha^2(x-s)^2)^2}$. We consider a homogeneous Dirichlet boundary condition at $\Gamma_D = \{0\}$ and a Neumann boundary condition at $\Gamma_N = \{1\}$, so that $u(0) = 0$ and $u'(1) = g := \frac{\alpha}{1+\alpha^2(1-s)^2}$.

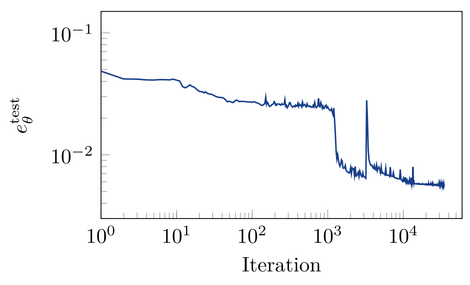
4.2.1.1. Non-parametric case. We fix the parameters $\boldsymbol{\sigma} = (\alpha, s) := (10, 0.5)$. Fig. 5(a) shows the evolution of the relative error during the r-adaptive optimization process for different numbers of elements, N, trained at a fixed LR of 10^{-2}. In all cases, the relative error stagnates fairly quickly, at about 300 iterations, with only marginal improvements when employing further iterations. This indicates a rather quick convergence towards a (possibly locally) optimal r-adaptive grid. The errors for the r-adapted mesh and uniform mesh, for different values of N, are displayed in Fig. 5(b), where the expected convergence rates are observed in both cases, but the r-adaptive method exhibits a better multiplicative constant than when considering the uniform mesh approach.

In particular, Fig. 6(a) and (b) compare FEM approximations using a uniform mesh versus an r-adaptive mesh using $N = 32$ elements. The mesh generated by the r-adaptive procedure clusters the nodes around the regions with rapid solution variations, ensuring a superior approximation. This demonstrates the method's capability to adjust the mesh density according to solution features during training.

4.2.1.2. Parametric case. We fix the number of elements to N and use the Ritz energy of the solution at a uniform mesh of size $h = \frac{1}{N}$ to compute the balanced Ritz functional in (12). To bias the NN towards high-gradient solutions associated with high values of α, we use 100 data points coming from a reversed base-2 logarithmic distribution, ranging from $\alpha_{\min} = 1$ to $\alpha_{\max} = 50$, i.e., $\alpha_j = \alpha_{\min} + \alpha_{\max} - 2^{\beta_j}$ where the β_j are equispaced in $[\log_2(\alpha_{\min}), \log_2(\alpha_{\max})]$. For the s parameter, we use 100 points equidistant in $[0.2, 0.8]$. Then, the 100^2 combinations of all (α, s) tuples were separated into training and test sets, as described in Section 4.1.2, ensuring the parameter 'corners', i.e., $(\alpha, s) \in \{(1, 0.2), (1, 0.8), (50, 0.2), (50, 0.8)\}$, were part of the training set. Thus, there were 7,000 tuples in the training set, with a batch of size 10 being processed per optimizer iteration, so that each epoch consisted of 700 iterations. The LR was set to 10^{-2} for 20 epochs, and then to 10^{-3} from then onward (until 50 epochs). The results of the training for $N = 16$ and $N = 256$ are shown in Fig. 7.



(a) Case $N = 16$.

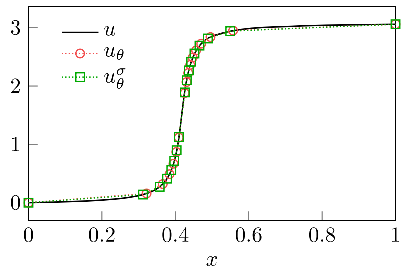


(b) Case $N = 256$.

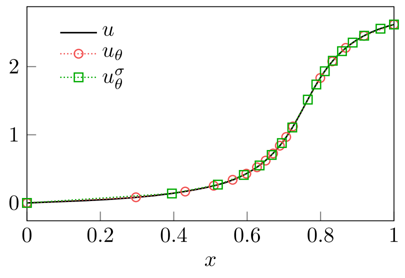
Fig. 7. Training histories during r-adaptivity for $N = 16$ and $N = 256$ in the parametric model problem (18a) with manufactured solution $u^\sigma(x) = \arctan(\alpha(x - s)) + \arctan(\alpha s)$ with $\boldsymbol{\sigma} = (\alpha, s)$.

Table 1
Relative error statistics for the uniform and the parametric r-adapted solutions associated with Fig. 7.

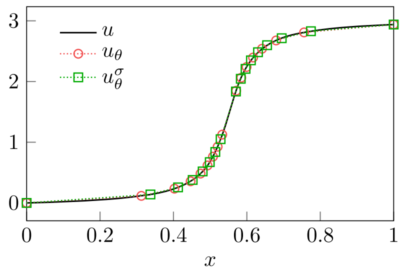
N	Dataset	r-adaptive mean e_θ^σ	r-adaptive max e_θ^σ	uniform grid mean e_h^σ	uniform grid max e_h^σ
16	Train	0.0838	0.0961	0.3770	0.5837
	Test	0.0838	0.0958	0.3777	0.5831
256	Train	0.0055	0.0061	0.0305	0.0398
	Test	0.0055	0.0061	0.0306	0.0398



(a) Test sample $\boldsymbol{\sigma} = (49.57, 0.42)$: $e_\theta \approx 0.0914$ and $e_\theta^\sigma \approx 0.0877$.



(b) Test sample $\boldsymbol{\sigma} = (19.88, 0.55)$: $e_\theta \approx 0.0803$ and $e_\theta^\sigma \approx 0.0819$.



(c) Test sample $\boldsymbol{\sigma} = (9.96, 0.76)$: $e_\theta \approx 0.0715$ and $e_\theta^\sigma \approx 0.0697$.

Fig. 8. Finite element solutions using a uniform and r-adapted mesh on three test samples for $N = 16$. In each case, we show the exact solution u, the solution u_θ coming from a case-by-case non-parametric r-adaptivity, and the solution u_θ^σ coming from the parametric r-adaptivity produced after the training in Fig. 7.

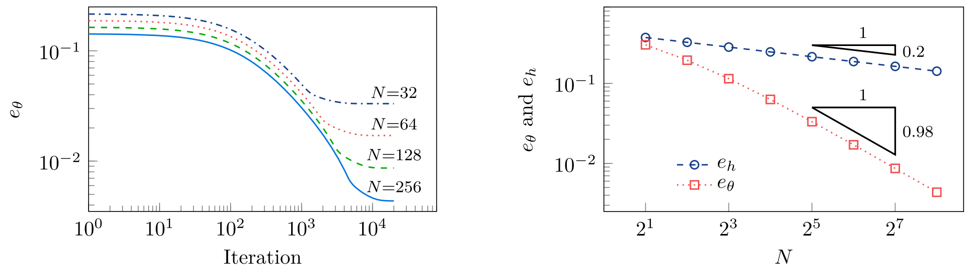
Table 1 compares the performance between the r-adaptive method and a uniform grid at different values of N. Focusing on the test set and $N = 16$, the r-adaptive approach achieves a mean relative error of 0.084 and a maximum relative error of 0.096. In contrast, the uniform grid results in significantly higher errors, with a mean of 0.38 and a maximum of 0.58. The training set exhibits similar improvements, and so do the errors for $N = 256$. These findings highlight the substantial accuracy improvements (approximately a factor of 5) provided by the parametric r-adaptive method over a standard uniform mesh across the parameter space. Furthermore, the consistent reduction of the relative errors observed on the test set confirms the NN's ability to effectively generate suitable adapted meshes, even for parameter combinations not encountered during training.

Fig. 8 provides qualitative results, displaying the solutions (u, u_θ and u_θ^σ) for three distinct test parameter pairs: $(\alpha, s) \in \{(49.27, 0.42), (19.88, 0.55), (9.96, 0.76)\}$. The visual results in Fig. 8 support that the parametric NN-generated meshes (see u_θ^σ) and the optimized non-parametric adaptive meshes (see u_θ) are comparable, with both effectively capturing the exact solution behavior. This confirms the approach's ability to handle parametric dependencies and adapt the discretization efficiently.

Curiously, we observed that in some cases the Ritz energy associated with u_θ was very similar, yet slightly higher (i.e., worse), than the one produced by u_θ^σ, despite the fact that u_θ corresponded to an r-adapted solution specially trained at a fixed parameter tuple. We believe this happened because the non-parametric network, which is initialized at a uniform mesh, gets stuck at a local minimum different from the global minimum as it r-adapts. Meanwhile, the parametric neural network appears to be more versatile in this respect, possibly because it is initialized differently and because it has a much wider landscape of $\boldsymbol{\sigma} = (\alpha, s)$ paths accessible to reach the global minimum as it is trained.

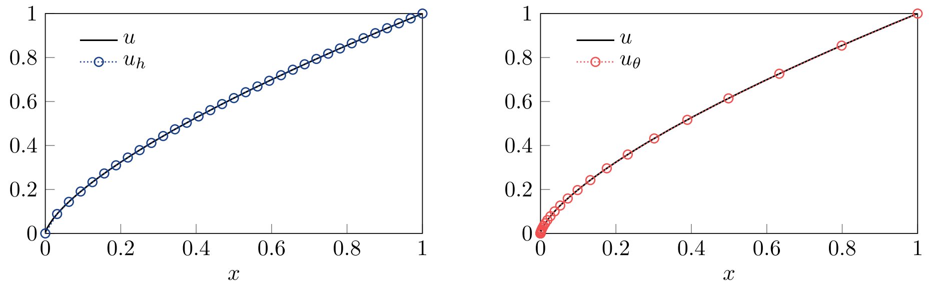
4.2.2. Singular power solutions

Consider the model problem (18a) with the manufactured solution $u(x) = x^\sigma$, and the boundary conditions set as $\Gamma_D = \{0\}$ and $\Gamma_N = \{1\}$, so that $-u''(x) = f(x) := \sigma(1 - \sigma)x^{\sigma-2}$ with $u(0) = 0$ and $u'(1) = \sigma$.



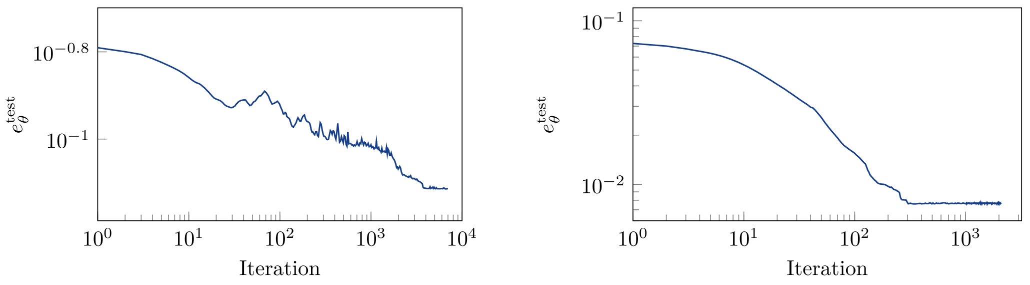
(a) Relative error during r-adaptive training. (b) Relative error convergence vs. N.

Fig. 9. Training histories during r-adaptive optimization for various mesh sizes N (left panel) and relative errors of the uniform and r-adapted finite element solutions as a function of N (right panel) for the non-parametric model problem (18a) with manufactured solution $u(x) = x^{0.7}$.



(a) Uniform mesh solution. (b) r-Adapted mesh solution.

Fig. 10. Finite element solutions using a uniform and r-adapted mesh according to the results in Fig. 9 for $N = 32$.



(a) Case $N = 10$. (b) Case $N = 256$.

Fig. 11. Training histories during r-adaptivity for $N = 10$ and $N = 256$ in the parametric model problem (18a) with manufactured solution $u^\sigma(x) = x^\sigma$.

4.2.2.1. Non-parametric case. We fix $\sigma = 0.7$, which leads to an exact solution that exhibits a singularity at Γ_D, where the derivative blows up. As before, Fig. 9(a) shows the relative error convergence during optimization, carried out for 20,000 iterations at a fixed LR of 10^{-2} for each of the values of N displayed. Fig. 9(b) shows the r-adapted relative error for each N along with the error for a uniform mesh of size $h = \frac{1}{N}$. From interpolation theory, we expect the convergence under uniform refinements to behave like $\mathcal{O}(h^{\min\{1,\sigma-0.5\}})$. Indeed, that is precisely the ratio observed in the plot, which exhibits a convergence rate of about 0.2. Meanwhile, the r-adaptive approach achieves a convergence rate of 0.98, significantly outperforming the uniform refinements, and being close to the theoretical optimal rate of 1 (see [51] and [52]).

Moreover, Fig. 10 illustrates the FEM approximations using a uniform and an r-adapted mesh with $N = 32$ elements. Notably, the r-adaptive solution demonstrates node concentration toward the singularity, aligning with the optimal distribution discussed in [51].

4.2.2.2. Parametric case. Consider 200 values of the parameter σ taken from a base-10 logarithmic distribution ranging from 0.51 to 5. The first, second, penultimate, and last points were purposely included in the training set, which consisted of 140 values of σ, so that 14 iterations comprised each epoch, since batches of size 10 were processed at each iteration. Fig. 11 shows the training history for meshes of sizes $N = 10$ and $N = 256$. For $N = 10$, we fixed the LR to 10^{-2} for 500 epochs (i.e., 7,000 iterations). For $N = 256$, we set an initial LR to 10^{-2}, which we reduced successively, every 3 epochs (i.e., 42 iterations), by 10^{-3}, until the LR attained a value of $3 \cdot 10^{-3}$; then, at 24 epochs (336 iterations), it was further adjusted to $5 \cdot 10^{-4}$, and at 30 epochs (420 iterations) it was decreased again to 10^{-4}, remaining fixed thereafter (until 150 epochs or 2,100 iterations).

Table 2
Relative error statistics for the uniform and the parametric r-adapted solutions associated with Fig. 11.

N	Dataset	r-adaptive mean e_θ^σ	r-adaptive max e_θ^σ	uniform grid mean e_h^σ	uniform grid max e_h^σ
10	Train	0.0811	0.7457	0.1437	0.9390
	Test	0.0752	0.5781	0.1357	0.8716
256	Train	0.0162	0.5658	0.0730	0.9090
	Test	0.0131	0.2908	0.0679	0.8120

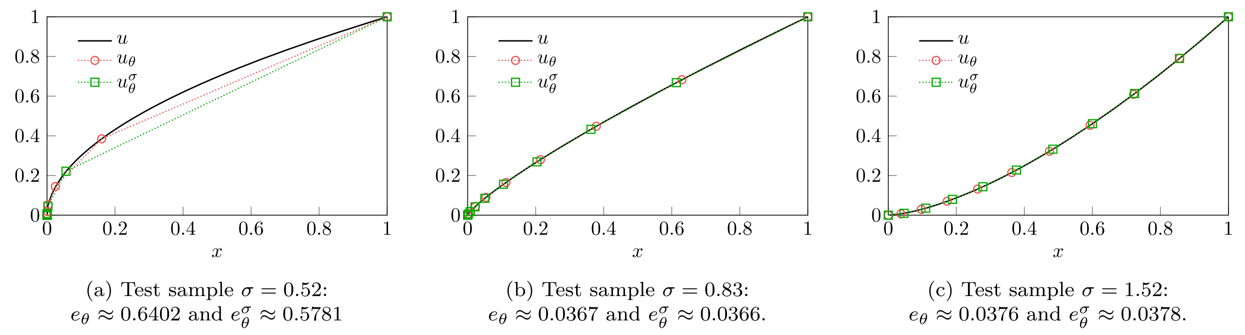


Fig. 12. Finite element solutions using a uniform and r-adapted mesh on three test samples for $N = 10$. In each case, we show the exact solution u, the solution u_θ coming from a case-by-case non-parametric r-adaptivity, and the solution u_θ^σ coming from the parametric r-adaptivity produced after the training in Fig. 11.

Statistics in Table 2 show that using a parametric r-adaptive method significantly improves accuracy compared to a uniform grid. At $N = 10$, for the test set, the adaptive meshes have a mean relative error of 0.075 and a maximum error of 0.58. In contrast, uniform meshes have a mean relative error of 0.14 and a maximum error of 0.87. The high maximum error in both cases is due to the singular solutions near $\sigma = 0.5$. Similar comparisons are seen in the training set. Much more substantial improvements are observed in the $N = 256$ case, in line with the results of Fig. 9. Overall, these results confirm that the NN effectively creates well-adapted meshes for new parameters, leading to better accuracy across the parameter space.

Fig. 12 shows the predictions of the parametric NN-based model (u_θ^σ) versus those of the non-parametric r-adaptive approach (u_θ) in three samples with parameters $\sigma \in \{0.52, 0.83, 1.52\}$. Both approaches show very good agreement, showcasing the efficacy of the NN-based parametric model. Note, however, that for $\sigma = 0.52$ the r-adapted solutions are clearly distinct, with u_θ^σ yielding a better approximation than the local minimizer u_θ, in line with the comments made in Section 4.2.1.

4.2.3. *Two-material transmission*

Now, for a positive constant $\sigma > 0$, we consider the other model problem (18b) with manufactured solution $u(x) = \sin(2\pi x)/\sigma(x)$, where the heterogeneous $\sigma(x)$ is a discontinuous function defined as $\sigma(x) = 1$ if $x < 0.5$ and $\sigma(x) = \sigma$ if $x \geq 0.5$, so that $-u''(x) = f(x) := 4\pi^2 \sin(2\pi x)$. The problem is solved with $\Gamma_D = \partial\Omega = \{0, 1\}$, so $u(0) = u(1) = 0$. Note that the solution is continuous since the discontinuity in $\sigma(x)$ occurs when $\sin(2\pi x)$ vanishes at $x = 0.5$. Thus, the (weak) gradient is well defined and $u \in H_0^1(\Omega)$. In addition, notice that given positive constants σ_1 and σ_2, if u is the solution with $\sigma = \frac{\sigma_2}{\sigma_1}$, then $\tilde{u} = \frac{1}{\sigma_1} u$ is the solution to the problem where the heterogeneous coefficient is instead given by $\tilde{\sigma}(x) = \sigma_1 \chi_{[0,0.5)}(x) + \sigma_2 \chi_{[0.5,1]}(x)$ (where $\chi_A(x)$ is the indicator function of a set $A \subset \Omega$).

4.2.3.1. *Non-parametric case.*

We fix the value $\sigma = 10$. The r-adaptive mesh incorporates the fixed point $x = 0.5$ to account for the material interface of the function $\sigma(x)$ (recall (4e)). As a result, the r-adaptive scheme allows for the movement of nodes across the fixed point, adapting to regions where the function changes significantly. Fig. 13 shows the relative error as the neural network is trained, for each N, for 10,000 iterations with a LR of 10^{-2}, as well as the error convergence for the r-adapted meshes and corresponding equispaced meshes. Although both display linear convergence, the r-adaptive method achieves a slightly smaller error magnitude, demonstrating its enhanced performance.

In addition, Fig. 14 shows the particular uniform and r-adapted solutions when $N = 32$. Notably, we observe that the r-adaptive mesh asymmetrically distributes 19 elements in $(0, 0.5)$ and 13 elements in $(0.5, 1)$, concentrating more elements in the region where the solution presents more variation.

4.2.3.2. *Parametric case.*

We use a base-10 logarithmic distribution for σ ranging from 10^{-4} to 10^4, consisting of 1,000 data points, i.e., σ is taken from 10^{β_j}, where the β_j are equispaced in $[-4, 4]$. The first, second, penultimate, and last points were explicitly included in the training set, which contained 700 values of σ, meaning each epoch consisted of 70 iterations (with a batch of size 10 processed per iteration). As in the non-parametric case, the node at $x = 0.5$ is fixed, so for a total of N elements we only have $N - 2$ movable nodes. Fig. 15 shows the training history for $N = 12$ and $N = 256$ elements, in both cases using a LR of 10^{-2} during the very first 30 epochs (2,100 iterations) and of 10^{-3} in the last 120 epochs (8,400 iterations).

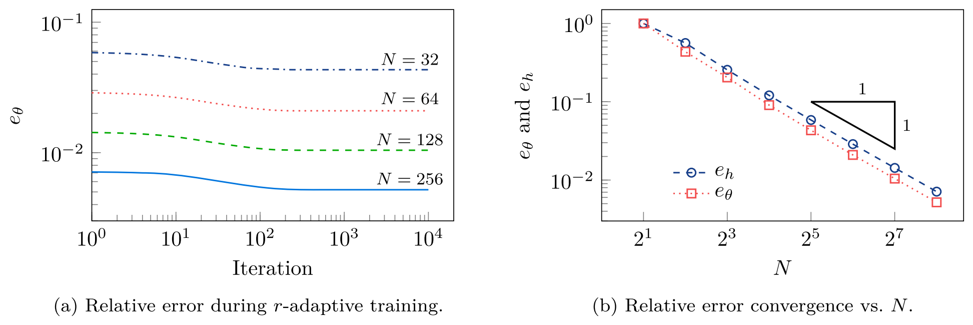


Fig. 13. Training histories during r-adaptive optimization for various mesh sizes N (left panel) and relative errors of the uniform and r-adapted finite element solutions as a function of N (right panel) for the non-parametric model problem (18b) with manufactured solution $u(x) = \sin(2\pi x)/\sigma(x)$ with $\sigma(x) = 1$ if $x < 0.5$ and $\sigma(x) = 10$ otherwise.

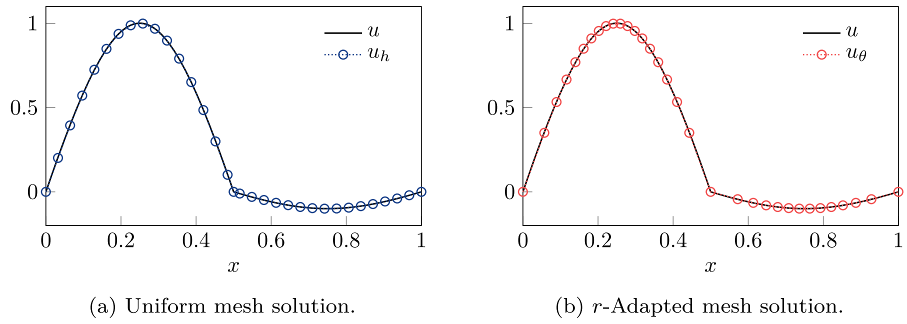


Fig. 14. Finite element solutions using a uniform and r-adapted mesh according to the results in Fig. 13 for $N = 32$.

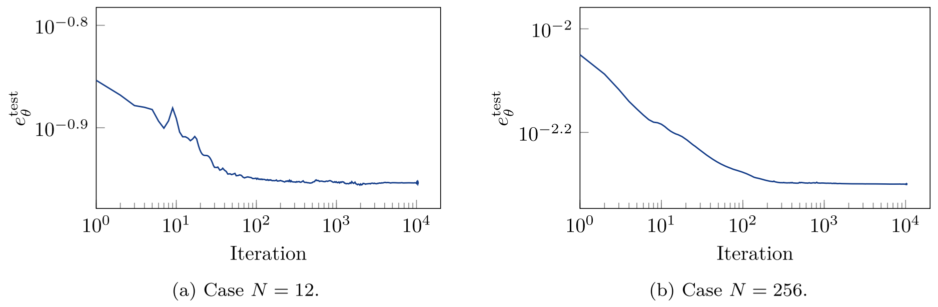


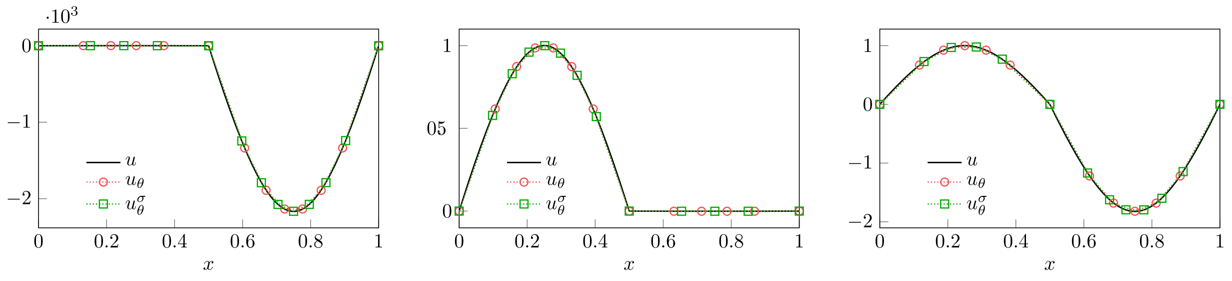
Fig. 15. Training histories during r-adaptivity for $N = 12$ and $N = 256$ in the parametric model problem (18b) with manufactured solution $u^\sigma(x) = \sin(2\pi x)/\sigma(x)$, where $\sigma(x) = 1$ if $x < 0.5$ and $\sigma(x) = \sigma$ if $x \geq 0.5$.

Table 3
Relative error statistics for the uniform and the parametric r-adapted solutions associated with Fig. 15.

N	Dataset	r-adaptive mean e_θ^σ	r-adaptive max e_θ^σ	uniform grid mean e_h^σ	uniform grid max e_h^σ
12	Train	0.1092	0.1521	0.1505	0.1505
	Test	0.1088	0.1521	0.1505	0.1505
256	Train	0.0050	0.0063	0.0071	0.0071
	Test	0.0050	0.0063	0.0071	0.0071

The results in Table 3 indicate that using a parametric r-adaptive method leads to slightly better accuracy than a uniform grid. Although both methods produce small errors, at $N = 12$, over the test set, the adaptive meshes achieve a mean relative error of 0.11, while the uniform meshes have a mean relative error of 0.15. Similar improvements were noted in the training set, and analogous results are observed for $N = 256$, indicating that the NN effectively generates well-adapted meshes, enhancing accuracy across different parameters.

Fig. 16 shows the predictions of the parametric NN-based model (u_θ^σ) versus those of the non-parametric r-adaptive approach (u_θ) in three samples with parameters $\sigma \in \{4.6 \cdot 10^{-4}, 8.9 \cdot 10^2, 5.5 \cdot 10^{-1}\}$. Even though both approaches lead to predictions that capture the exact solution (and are symmetric in each half-domain), note that the r-adaptive meshes are different, since at each half they



(a) Test sample $\sigma = 4.6 \cdot 10^{-4}$: $e_\theta \approx 0.1140$ and $e_\theta^\sigma \approx 0.0994$.

(b) Test sample $\sigma = 8.9 \cdot 10^{2}$: $e_\theta \approx 0.1140$ and $e_\theta^\sigma \approx 0.0995$.

(c) Test sample $\sigma = 5.5 \cdot 10^{-1}$. $e_\theta \approx 0.1337$ and $e_\theta^\sigma \approx 0.1334$.

Fig. 16. Finite element solutions using a uniform and r-adapted mesh on three test samples for $N = 12$. In each case, we show the exact solution u, the solution u_θ coming from a case-by-case non-parametric r-adaptivity, and the solution u_θ^σ coming from the parametric r-adaptivity produced after the training in Fig. 15.

contain a different number of nodes. As pointed out before, it turns out the solutions coming from the parametric NN actually yield better results (i.e., lower errors) than those from the non-parametric NN trained at the fixed σ. The reasons are likely the ones stated previously. Having said that, we further experimented with the non-parametric NN, modifying its optimization procedure, so instead of using Adam, we utilized stochastic gradient descent with a learning rate of 0.01, momentum of 0.95, and Nesterov acceleration. By doing so, we were able to avoid the sub-optimal local minima and match the results of the parametric NN for the first two cases in Fig. 16, but retained the same (locally minimizing) solution in the last case.

4.3. *Two-dimensional experiments*

We present two two-dimensional benchmark problems, in both their non-parametric and parametric versions. Quadrilateral tensor-product meshes are used to discretize the domains. In the first example, we consider the domain $\Omega = (0,1)^2$, whilst in the second, Ω is an L-shape domain. The Adam optimizer is used in all of our experiments, both in the non-parametric and parametric cases, with the LR varying on a case-by-case basis. Lastly, in the non-parametric examples the linear systems are solved using `jax.experimental.sparse.linalg` within the JAX AD framework, whereas in the parametric examples they are solved, with batches of size 1, via `scipy.sparse.linalg.spsolve`, avoiding differentiation with respect to the solution.

4.3.1. *Arctangent sigmoid solutions*

The two-dimensional extension of the one-dimensional arctangent sigmoid solution from Subsection 4.2.1 is obtained by setting $\Omega = (0,1)^2$ and considering the Poisson problem (18a) with manufactured solution $u(x,y) = u_1(x)u_2(y)$, where $u_j(t) = \arctan(\alpha(t - s_j)) + \arctan(\alpha s_j)$ for $j = 1,2$. Homogeneous Dirichlet boundary conditions are established at the left and bottom boundaries (i.e. $x = 0$ or $y = 0$), while Neumann boundary conditions are set at the right and top boundaries (i.e. $x = 1$ or $y = 1$). The function $f = -\Delta u$ can thus be analytically calculated to be the expression

$$f(x,y) = u_1''(x)u_2(y) + u_1(x)u_2''(y) \qquad \text{where} \qquad u_j''(t) = \frac{2\alpha^3(t - s_j)}{(1 + (\alpha(t - s_j))^2)^2}\,, \quad j = 1,2\,. \tag{23}$$

The integrals involving f are approximated numerically using 50^2 Gauss-Legendre quadrature points per element, since the resulting expressions are challenging to integrate exactly. The high number of quadrature points is important to avoid the issues described in Section 2.3.

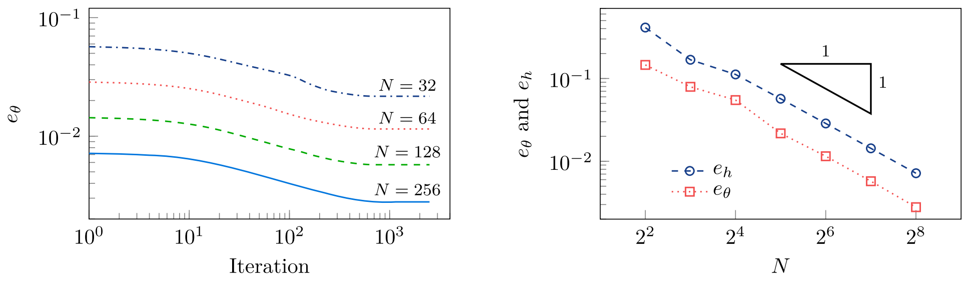
4.3.1.1. *Non-parametric case.*

Let $\alpha = 10$ and $s_1 = s_2 = 0.05$. For each value of N, corresponding to the number of elements per side, we optimize the r-adaptive mesh during 2,500 iterations using the Adam optimizer with a fixed LR of 10^{-2}, with the results being those of Fig. 17(a). Fig. 17(b), like Fig. 8(b), shows the r-adapted and uniform-mesh (with $h = \frac{1}{N}$) solutions converge linearly, with the r-adapted solutions exhibiting a better constant.

Fig. 18 shows the uniform and r-adapted predictions featuring $N = 32$ elements per side for a total of 32^2 elements. As expected, the r-adaptive method focuses nodes in areas with sharp gradients, thereby enhancing the accuracy of the approximation.

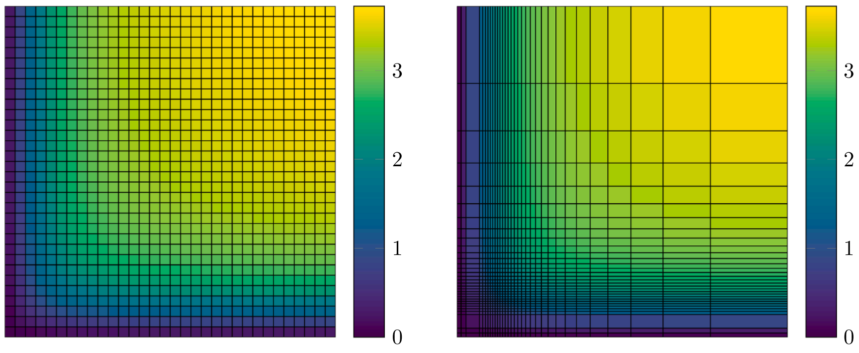
4.3.1.2. *Parametric case.*

Let the parameter be $\boldsymbol{\sigma} = (\alpha, s_1, s_2)$. We sample α from the same inverted base-2 logarithmic distribution described in Section 4.2.1, ranging from 20 to 1 with 20 data points, whereas s_1 and s_2 are uniformly distributed in $[0.1, 0.9]$ with 10 data points each. Thus, the training set consisted of 1,400 $\boldsymbol{\sigma}$-tuples, with each epoch comprised of 1,400 iterations (using a batch size of 1). Fig. 19 shows the training evolution with $N = 9$ and $N = 64$ elements per side. For $N = 9$, training is carried out for 1,000 epochs (1,400,000 iterations) at an initial LR of 10^{-2} for the first 150 epochs (210,000 iterations) and 10^{-3} thereafter, while for $N = 64$ it is trained for 500 epochs (700,000 iterations) with a LR of 10^{-2} for the first 20 epochs (28,000 iterations) and 10^{-3} afterward.



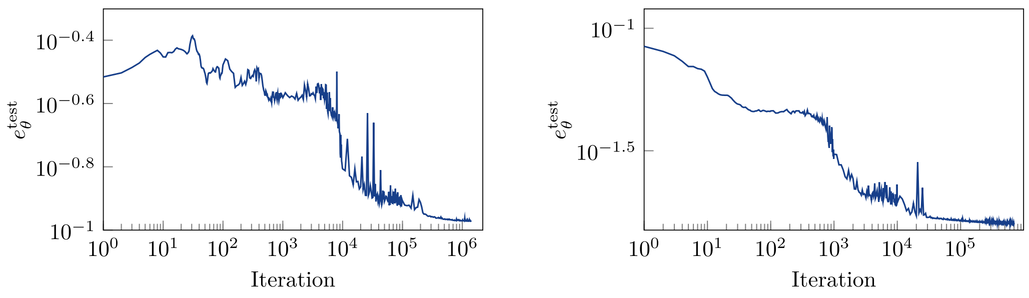
(a) Relative error during r-adaptive training. (b) Relative error convergence vs. N.

Fig. 17. Training histories during r-adaptive optimization for each value of N (left panel), and relative errors of the uniform and r-adapted finite element solutions as a function of N (right panel) for the non-parametric model problem (18a) with manufactured solution $u(x, y) = u_1(x)u_2(y)$ where $u_j(t) = \arctan(10t - 0.05) + \arctan(0.5)$ for $j = 1, 2$.



(a) Uniform mesh solution. (b) r-Adapted mesh solution.

Fig. 18. Finite element solutions using a uniform and r-adapted mesh according to the results in Fig. 17 for $N = 32$.



(a) Case $N = 9$. (b) Case $N = 64$.

Fig. 19. Training histories during r-adaptivity for $N = 9$ and $N = 64$ elements per side in the parametric model problem (18a), with manufactured solution $u(x, y) = u_1(x)u_2(y)$, where $u_j(t) = \arctan(\alpha(t - s_j)) + \arctan(\alpha s_j)$ for $j = 1, 2$.

Table 4 compares the relative errors of the approximation using NN-based r-adaptive meshes and uniform meshes.

For $N = 64$ and the test set, the r-adaptive method gives a mean relative error of 0.017 and a maximum relative error of 0.022. This outperforms the results on the uniform grid, where the mean and maximum relative errors are 0.046 and 0.064, respectively. Similar numbers are obtained for the training set. The results confirm the findings in Table 1, namely, the NN-based meshes result in an improvement compared to uniform meshes.

Fig. 20 displays the solutions u_θ and u_θ^σ for selected parameter values, showing relatively good agreement between them and their respective two-dimensional non-parametric and parametric meshes.

Table 4
Relative error statistics for the uniform and the parametric r-adapted solutions associated with Fig. 19.

N	Dataset	r-adaptive mean e_θ^σ	r-adaptive max e_θ^σ	uniform grid mean e_h^σ	uniform grid max e_h^σ
9	Train	0.1168	0.1422	0.2842	0.4578
	Test	0.1163	0.1424	0.2879	0.4548
64	Train	0.0173	0.0215	0.0457	0.0636
	Test	0.0172	0.0216	0.0460	0.0636

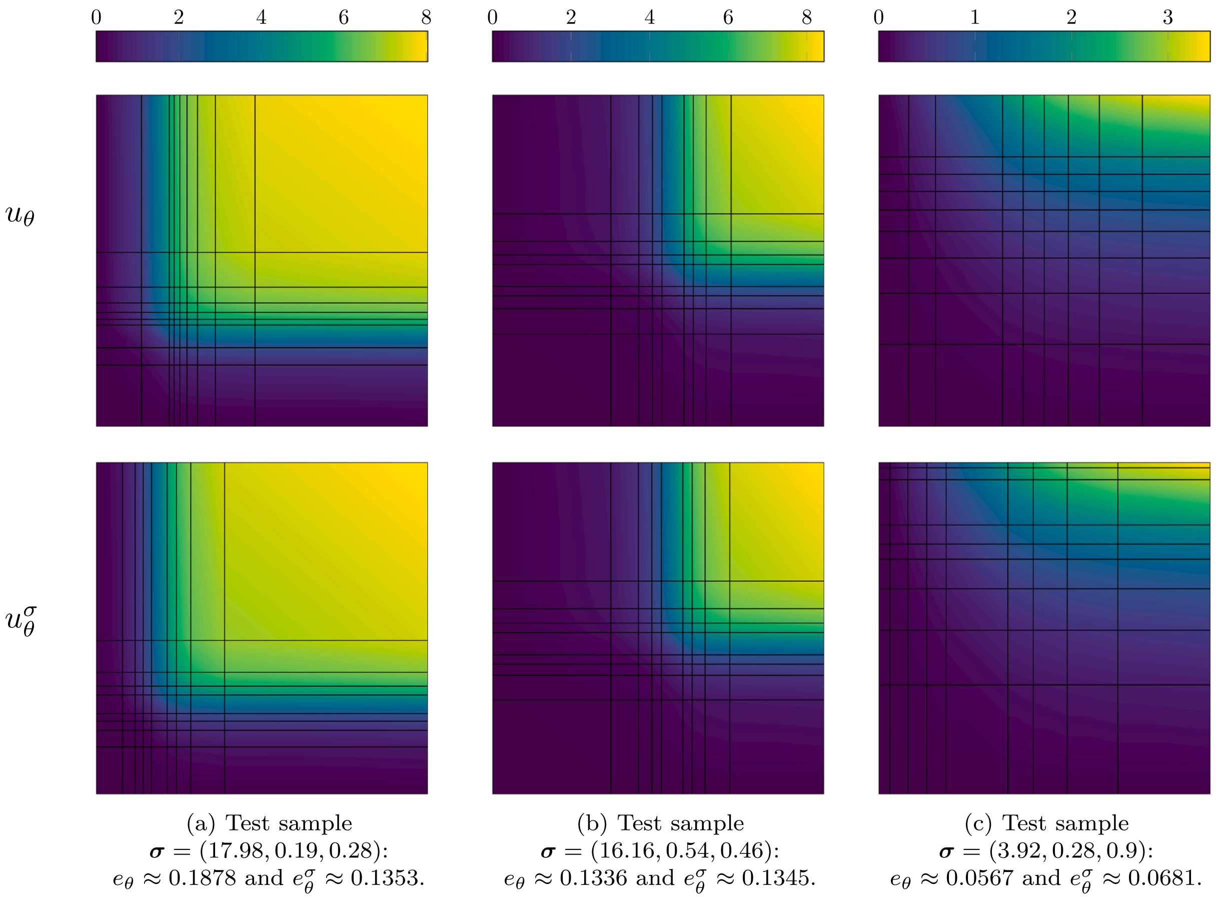


Fig. 20. Finite element solutions using a uniform and r-adapted mesh on three test samples for $N = 9$. In each case, we show the solution u_θ coming from a case-by-case non-parametric r-adaptivity, and the solution u_θ^σ coming from the parametric r-adaptivity produced after the training in Fig. 19.

4.3.2. Multi-material L-shape domain

We consider the benchmark problem (18a) on an L-shape domain, $\Omega = (0,1)^2 \setminus \{(0.5,1)\times(0,0.5)\}$, with $f = 1$ and homogeneous Dirichlet boundary conditions, i.e. $u|_{\partial\Omega} = 0$. Given positive constants σ_1 and σ_2, the heterogeneous $\sigma(\boldsymbol{x})$ is given by

$$\sigma(\boldsymbol{x}) = \chi_{(0,0.5)\times(0.5,1)}(\boldsymbol{x}) + \sigma_1\chi_{(0,0.5)^2}(\boldsymbol{x}) + \sigma_2\chi_{(0.5,1)^2}(\boldsymbol{x}) = \begin{cases} \sigma_0 = 1 & \boldsymbol{x} \in (0,0.5)\times(0.5,1), \\ \sigma_1 & \boldsymbol{x} \in (0,0.5)^2, \\ \sigma_2 & \boldsymbol{x} \in (0.5,1)^2, \end{cases} \tag{24}$$

where $\chi_A(\boldsymbol{x})$ is the indicator function for the set $A \subset \Omega$. Note that given a triplet of positive numbers $(\tilde{\sigma}_0, \tilde{\sigma}_1, \tilde{\sigma}_2)$, if u is the solution of the problem above with $\sigma_1 = \frac{\tilde{\sigma}_1}{\tilde{\sigma}_0}$ and $\sigma_2 = \frac{\tilde{\sigma}_2}{\tilde{\sigma}_0}$, then $\tilde{u} = \frac{1}{\tilde{\sigma}_0}u$ is the solution of the model problem for a general three-material $\tilde{\sigma}(\boldsymbol{x})$ given by

$$\tilde{\sigma}(\boldsymbol{x}) = \tilde{\sigma}_0\sigma(\boldsymbol{x}) = \tilde{\sigma}_0\chi_{(0,0.5)\times(0.5,1)}(\boldsymbol{x}) + \tilde{\sigma}_1\chi_{(0,0.5)^2}(\boldsymbol{x}) + \tilde{\sigma}_2\chi_{(0.5,1)^2}(\boldsymbol{x}).$$

Thus, with only two parameters, i.e., $\boldsymbol{\sigma} = (\sigma_1, \sigma_2)$, one is able to characterize the three-material case.

With this f and boundary conditions, the solutions are not known analytically, even for the simplest $\boldsymbol{\sigma} = (\sigma_1, \sigma_2)$. Nevertheless, they are known to have a singular gradient at the re-entrant corner, and the 'intensity' of this singularity varies with $\boldsymbol{\sigma}$. Indeed, in the limit $\sigma_2 \to \infty$, leaving σ_1 constant, the singularity will disappear entirely, effectively leading to the equation being satisfied in a

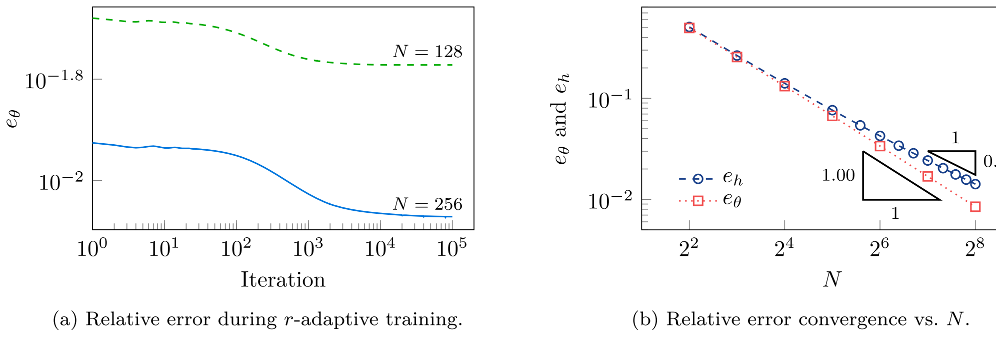


Fig. 21. Training histories during r-adaptive optimization for each value of N (left panel), and relative errors of the uniform and r-adapted finite element solutions as a function of N (right panel) for the non-parametric model problem $\Delta u = 1$ in the L-shape domain with $u|_{\partial\Omega} = 0$.

rectangle. As a replacement for the exact solutions, we resort to highly accurate approximations of their Ritz energy computed in carefully refined triangular meshes with the open source FEM software Netgen/NGSolve[53,54]. Note that this 'exact' Ritz energy is enough to compute the relevant $\|\cdot\|_b$ relative errors in Section 4.1.3 (rather than interpolate between meshes).

Regarding our implementation, we use quadrilateral meshes with N elements per side, where, to ensure the presence of the re-entrant corner in the mesh, nodes with $x = 0.5$ and $y = 0.5$ coordinates are fixed. To emulate the L-shape domain, the nodes in the bottom right quadrant are tagged as zero Dirichlet nodes, and these are dynamically labeled after each r-adaptive step, as described in Section 3.2. Due to the simplicity of the forcing and boundary conditions (i.e., $f = 1$ and $u|_{\partial\Omega} = 0$), all integrals in the finite element variational formulation are computed exactly using numerical quadrature (with 2^2 Gauss-Legendre quadrature points).

4.3.2.1. Non-parametric case. Let $\sigma = (1,1)$, so that we solve the Poisson equation $\Delta u = 1$ with $u|_{\partial\Omega} = 0$. This is *not* the usual L-shape domain example where the solution (i.e., $u(r,\theta) = r^{2/3}\sin(\frac{2}{3}\theta)$ solving $\Delta u = 0$) is in $H^{1+s-\delta}(\Omega)$ for $s = \frac{2}{3}$ and $\delta > 0$. However, it is still a classical example of a 'nice' second-order operator with smooth right-hand side and boundary conditions that does not lead to a smooth solution. Indeed, due to the nonconvex re-entrant corner, the solution has a singular gradient and does not belong to $C^1(\Omega)$. Instead, it is in $H^{1+s}(\Omega)$ where s appears to be around 0.72 as computed with Netgen/NGSolve under high-order uniform refinements. The Ritz energy used as a reference for the relative error computations is $\mathcal{J}(u) = -0.00668986$, which was computed with order-five polynomials on an unstructured triangular mesh specially refined at the re-entrant corner using Netgen/NGSolve.

For each given (even) number of elements per side, N, note that the number of 'active' elements in a uniform grid of size $h = \frac{1}{N}$ is actually $\frac{3}{4}N^2$ and the number of degrees of freedom is $\frac{3}{4}N^2 - 2N + 1$. At different N, we optimize the r-adaptive mesh during 100,000 iterations using the Adam optimizer with a fixed LR of 10^{-2}, as shown in Fig. 21(a). The convergence in N of the uniform mesh and r-adapted solutions is shown in Fig. 21(b), where the corresponding number of degrees of freedom is also $\frac{3}{4}N^2 - 2N + 1$, because, in all cases observed, the nodes never crossed the midpoint. We observe that the convergence rate with respect to $N = \frac{1}{h}$ under uniform refinements at $N = 2^8$ is around 0.78 and appears to be decreasing (indeed, it will eventually decrease to around 0.72). Remarkably, in the r-adaptive case, the mean convergence rate is 1.00, which is the best possible asymptotic convergence rate (see [52]).

Fig. 22 displays the uniform and r-adapted solutions for $N = 32$ elements per side. The r-adaptive mesh is refining strongly toward the re-entrant corner and moderately toward the exterior boundaries, and the scale of the gradient magnitude clearly shows that it captures the singularity more accurately than the uniform mesh.

4.3.2.2. Parametric case. Let the parameter be $\sigma = (\sigma_1, \sigma_2)$ and sample 20 values of σ_1 and 20 values of σ_2 uniformly from a base-10 logarithmic distribution from 10^{-1} to 10^1, as in Section 4.2.3, leading to 20^2 parameter tuples, and a training set of 280 σ-tuples, so that, with a batch size of 1, each epoch consisted of 280 iterations. The reference Ritz energy (which acts as the 'exact' value) for every σ was computed with Netgen/NGSolve using piecewise-linear functions on a fine unstructured triangular mesh. Fig. 23 shows the parametric NN training on the $N = 10$ and $N = 64$ cases, though we note that the number of 'active' elements is variable due to the bottom-right quadrant being removed. For $N = 10$, training is carried out for 5,000 epochs (1,400,000 iterations) with a fixed LR of 10^{-2}. Meanwhile, for $N = 64$, we train using Adam with an initial LR of 10^{-2} for 50 epochs (14,000 iterations), followed by a LR of 10^{-3} for the subsequent 300 epochs (84,000 iterations), and further decreased to $5 \cdot 10^{-5}$ thereafter, for a total of 12,500 epochs (3,500,000 iterations).

Table 5 shows the error statistics resulting from the r-adaptive parametric NN and from uniform meshes. For the test set with $N = 64$, the r-adaptive method gives a mean relative error of 0.032 and a maximum relative error of 0.054, which is better than the 0.045 and 0.103 obtained with uniform grids, respectively. Similar improvements are seen for the training set and for $N = 10$. Thus, the parametric NN generates well-adapted meshes, enhancing accuracy across different parameters in this challenging two-dimensional scenario.

Lastly, for $N = 10$, Fig. 24 compares the performance of the parametric NN with that of non-parametric NNs on some samples of the test set. Each non-parametric network is trained for 15,000 epochs using a fixed learning rate of 10^{-2}. In line with other

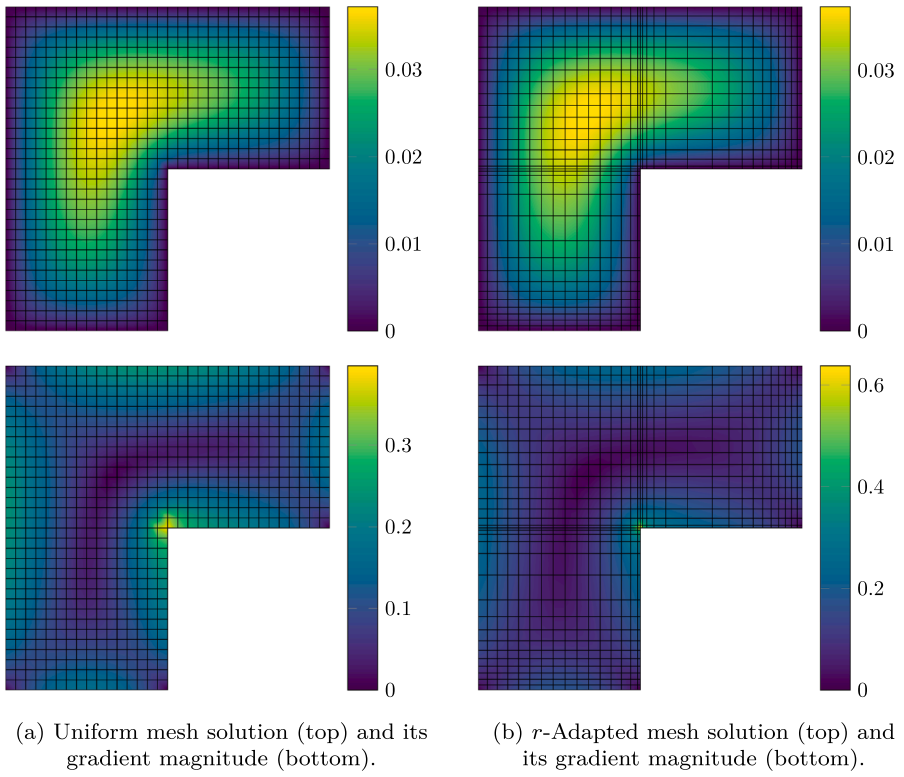


Fig. 22. Finite element solutions using a uniform and r-adapted mesh according to the results in Fig. 21 for $N = 32$.

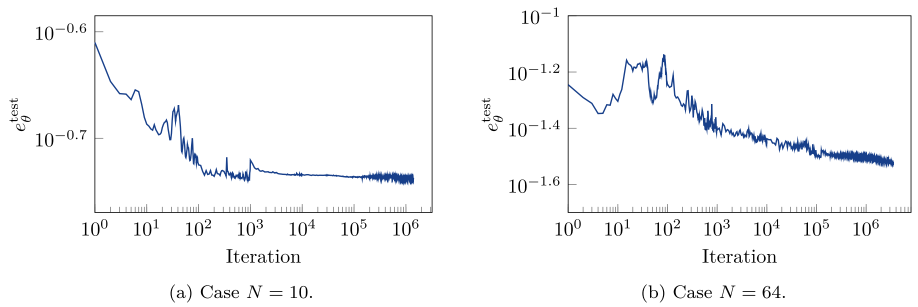
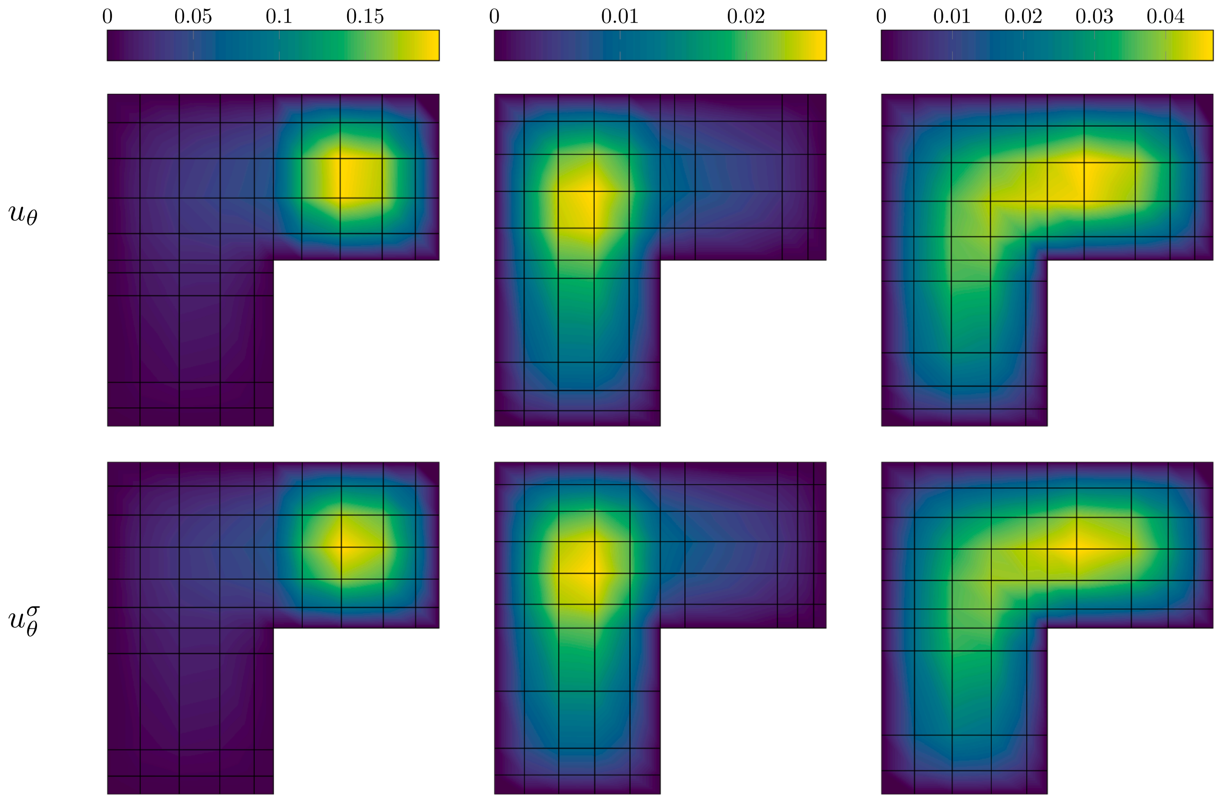


Fig. 23. Training histories during r-adaptivity for $N = 10$ and $N = 64$ elements per side in the parametric model problem (18b) solving $\nabla \cdot (\sigma(\boldsymbol{x})\nabla u) = 1$ and $u|_{\partial\Omega} = 0$ on a multi-material L-shape domain with $\sigma(\boldsymbol{x})$ defined by (24).

Table 5
Relative error statistics for the uniform and the parametric r-adapted solutions associated with Fig. 23.

N	Dataset	r-adaptive mean e_θ^σ	r-adaptive max e_θ^σ	uniform grid mean e_h^σ	uniform grid max e_h^σ
10	Train	0.1933	0.2530	0.3149	0.5029
	Test	0.1933	0.2510	0.3191	0.5005
64	Train	0.0315	0.0598	0.0458	0.1086
	Test	0.0317	0.0541	0.0453	0.1027

examples above, and likely for the same reasons, the results coming from the parametric NN yield better results (lower values of Ritz energy) than the non-parametric NN trained at the specific value of $\boldsymbol{\sigma}$. This reflects what appears to be an added (and somewhat counterintuitive) benefit of having a parametric NN for a class of problems, rather than a NN specialized to a specific problem.



(a) Test sample $\boldsymbol{\sigma} = (2.34, 0.1)$: $e_\theta \approx 0.2193$ and $e_\theta^\sigma \approx 0.2157$.
(b) Test sample $\boldsymbol{\sigma} = (1.83, 4.83)$: $e_\theta \approx 0.1979$ and $e_\theta^\sigma \approx 0.1699$.
(c) Test sample $\boldsymbol{\sigma} = (0.89, 0.55)$: $e_\theta \approx 0.2147$ and $e_\theta^\sigma \approx 0.1906$.

Fig. 24. Finite element solutions using a uniform and *r*-adapted mesh on three test samples for $N = 10$. In each case, we show the solution u_θ coming from a case-by-case non-parametric *r*-adaptivity, and the solution u_θ^σ coming from the parametric *r*-adaptivity produced during optimization of Fig. 23.

5. Conclusions and future work

This work presents a novel *r*-adaptive FEM tailored to solve parametric PDEs using NNs. The core of the proposed approach involves using an NN to generate an optimized, parameter-dependent mesh, which is then used by a standard FEM solver to compute the solution for each parameter instance. This strategy combines the robustness and reliability guarantees inherent to the FEM with the capabilities of NNs for adaptive node relocation. By concentrating mesh elements in regions with high-variation solutions, the method achieves significant accuracy improvements compared to uniform meshes, particularly for problems exhibiting sharp gradients, boundary layers, or singularities. The *r*-adaptive method approaches the convergence rate associated with smooth solutions, even when the solution is singular. The method was implemented on top of JAX to benefit from its automatic differentiation and accelerated linear algebra via just-in-time compilation, but the linear solve associated with the FEM can be implemented outside of this framework with any desirable solver, thus maximizing performance.

The effectiveness of the NN-based *r*-adaptive FEM approach is demonstrated through numerical experiments on one- and two-dimensional parametric Poisson problems, showcasing its ability to robustly handle varying parameters in both the diffusion coefficient and the source term. Despite having to initially train a NN, the flexibility of being able to solve a parametric PDE constitutes a great benefit when compared with more traditional *r*-adaptive methods. The training of the NN relies on minimizing the Ritz energy, which currently restricts the method's applicability to symmetric and coercive problems, such as those arising from elliptic self-adjoint problems. Furthermore, the current implementation focuses on first-order piecewise-polynomial elements and tensor-product meshes.

Future research will focus on extending its applicability to non-symmetric or indefinite PDEs, potentially by modifying the loss function using first-order system least-squares formulations (FOSLS). Another important avenue worth exploring is the implementation of high-order polynomials, irregular geometries, and different element shapes (e.g., simplices). If achievable, this will require the development of more sophisticated mesh generation strategies that remain compatible with the automatic differentiation required by NN training, while also avoiding harmful mesh degeneracy. Moreover, we would like to investigate the integration of goal-oriented adaptive schemes to further refine the mesh optimization process based on specific quantities of interest. Finally, we believe it is critical to develop all these improvements with the aim of eventually addressing high-dimensional problems in the PDE-parameter space.

CRediT authorship contribution statement

Danilo Aballay: Visualization, Validation, Software, Data curation; **Federico Fuentes:** Writing – review & editing, Writing – original draft, Supervision, Software, Resources, Project administration, Funding acquisition, Formal analysis, Conceptualization; **Vicente Iligaray:** Visualization, Validation, Software, Data curation; **Ángel J. Omella:** Visualization, Validation, Supervision, Software, Methodology, Investigation, Data curation, Conceptualization; **David Pardo:** Writing – review & editing, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Manuel A. Sánchez:** Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Data curation, Conceptualization; **Ignacio Tapia:** Visualization, Validation, Software, Data curation; **Carlos Uriarte:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Data availability

The code for reproducing the results is dully referenced in the manuscript and publicly available in Github.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

AJO, CU and DP declare this work has received funding from the following Research Projects/Grants: European Union's Horizon Europe research and innovation programme under the Marie Sklodowska-Curie Action MSCA-DN-101119556 (IN-DEEP); PID2023-146678OB-I00 funded by MICIU/AEI/10.13039/501100011033 and by FEDER, EU; PID2023-146668OA-I00 funded by MICIU/AEI/10.13039/501100011033 and by FEDER, EU; BCAM Severo Ochoa accreditation of excellence CEX2021-001142-S funded by MICIU/AEI/10.13039/501100011033; Basque Government through the BERC 2022–2025 program; BEREZ-IA (KK-2023/00012) and RUL-ET(KK-2024/00086), funded by the Basque Government through ELKARTEK; Consolidated Research Group MATHMODE (IT1456-22) given by the Department of Education of the Basque Government; BCAM-IKUR-UPV/EHU, funded by the Basque Government IKUR Strategy and by the European Union NextGenerationEU/PRTR. DA, VI, IT, FF and MS were partially supported by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID based in Chile. DA, VI and FF were partially supported by the Office of Naval Research (ONR) award N629092312098, and MS was partially funded by FONDECYT Regular (ANID), grant N. 1221189.Lastly, all authors acknowledge support by grant N. FOVI240119 (Fomento a la Vinculación Internacional para instituciones de Investigación), ANID Chile.

References

[1] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, SIAM, 2002.
[2] T.J.R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Prentice-Hall, 1987. Reprinted by Dover Publications, 2000.
[3] O.C. Zienkiewicz, R.L. Taylor, J.Z. Zhu, The Finite Element Method: Its Basis and Fundamentals, Elsevier, 2005.
[4] S.C. Brenner, The Mathematical Theory of Finite Element Methods, Springer, 2008.
[5] I. Babuška, M. Suri, The p and h-p versions of the finite element method, basic principles and properties, SIAM Rev. 36 (4) (1994) 578–632.
[6] C. Schwab, M. Suri, The p and hp versions of the finite element method for problems with boundary layers, Math. Comput. 65 (216) (1996) 1403–1429.
[7] M. Ainsworth, J.T. Oden, A posteriori error estimation in finite element analysis, Comput. Methods Appl. Mech. Eng. 142 (1–2) (1997) 1–88.
[8] L. Demkowicz, W. Rachowicz, P. Devloo, A fully automatic hp-adaptivity, J. Sci. Comput. 17 (2002) 117–142.
[9] W. Rachowicz, D. Pardo, L. Demkowicz, Fully automatic hp-adaptivity in three dimensions, Comput. Methods Appl. Mech. Eng. 195 (37–40) (2006) 4816–4842.
[10] D. Pardo, Multigoal-oriented adaptivity for hp-Finite element methods, Procedia Comput. Sci. 1 (1) (2010) 1953–1961.
[11] C. Carstensen, M. Feischl, M. Page, D. Praetorius, Axioms of adaptivity, Comput. Math. Appl. 67 (6) (2014) 1195–1253.
[12] C.J. Budd, W. Huang, R.D. Russell, Adaptivity with moving grids, Acta Numer. 18 (2009) 111–241.
[13] W. Huang, R.D. Russell, Adaptive Moving Mesh Methods, 174, Springer Science & Business Media, 2010.
[14] M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.
[15] J. Blechschmidt, O.G. Ernst, Three ways to solve partial differential equations with neural networks — a review, GAMM-Mitteilungen 44 (2) (2021) e202100006.
[16] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[17] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, (2019). arXiv preprint arXiv:1912.00873.
[18] E. Kharazmi, Z. Zhang, G.E. Karniadakis, hp-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547.
[19] S. Rojas, P. Maczuga, J. Muñoz-Matute, D. Pardo, M. Paszyński, Robust variational physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 425 (2024) 116904.
[20] C. Uriarte, M. Bastidas, D. Pardo, J.M. Taylor, S. Rojas, Optimizing variational physics-Informed neural networks using least squares, Comput. Math. Appl. 185 (2025) 76–93.
[21] B. Yu, The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12.
[22] Y. Liao, P. Ming, Deep nitsche method: deep ritz method with essential boundary conditions, (2019). arXiv preprint arXiv:1912.01309.

[23] C. Uriarte, D. Pardo, I. Muga, J. Muñoz-Matute, A deep double ritz method (D2RM) for solving partial differential equations using neural networks, Comput. Methods Appl. Mech. Eng. 405 (2023) 115892.
[24] J.A.A. Opschoor, C. Schwab, C. Xenophontos, Neural networks for singular perturbations, (2024). arXiv preprint arXiv:2401.06656.
[25] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, 1, MIT Press Cambridge, 2016.
[26] P. Petersen, M. Raslan, F. Voigtlaender, Topological properties of the set of functions generated by neural networks of fixed size, Found. Comput. Math. 21 (2021) 375–444.
[27] C. Uriarte, Solving partial differential equations using artificial neural networks, (2024). arXiv preprint arXiv:2403.09001.
[28] S. Berrone, C. Canuto, M. Pintore, Variational physics informed neural networks: the role of quadratures and test functions, J. Sci. Comput. 92 (3) (2022) 100.
[29] J.A. Rivera, J.M. Taylor, Á.J. Omella, D. Pardo, On quadrature rules for solving partial differential equations using neural networks, Comput. Methods Appl. Mech. Eng. 393 (2022) 114710.
[30] C. Uriarte, J.M. Taylor, D. Pardo, O.A. Rodríguez, P. Vega, Memory-based Monte Carlo integration for solving partial differential equations using neural networks, in: International Conference on Computational Science, Springer, 2023, pp. 509–516.
[31] S. Pal, K. Azizzadenesheli, V. Singh, Solving PDEs via Learnable Quadrature, 2025, (OpenReview). https://openreview.net/forum?id=tl63stKeSC.
[32] J.M. Taylor, D. Pardo, Stochastic quadrature rules for solving PDEs using neural networks, (2025). arXiv preprint 2504.11976.
[33] Á.J. Omella, D. Pardo, *r*-Adaptive deep learning method for solving partial differential equations, Comput. Math. Appl. 153 (2024) 33–42.
[34] S. Baharlouei, J.M. Taylor, C. Uriarte, D. Pardo, A least-squares-based neural network (LS-Net) for solving linear parametric PDEs, Comput. Methods Appl. Mech. Eng. 437 (2025) 117757. https://doi.org/10.1016/j.cma.2025.117757
[35] S. Badia, W. Li, A.F. Martín, Finite element interpolated neural networks for solving forward and inverse problems, Comput. Methods Appl. Mech. Eng. 418 (2024) 116505.
[36] S. Badia, W. Li, A.F. Martín, Adaptive finite element interpolated neural networks, Comput. Methods Appl. Mech. Eng. 437 (2025) 117806.
[37] S. Badia, W. Li, A.F. Martín, Compatible finite element interpolated neural networks, Comput. Methods Appl. Mech. Eng. 439 (2025) 117889.
[38] C. Uriarte, D. Pardo, Á.J. Omella, A finite element based deep learning solver for parametric PDEs, Comput. Methods Appl. Mech. Eng. 391 (2022) 114562.
[39] J.Y. Lee, S. Ko, Y. Hong, Finite element operator network for solving elliptic-Type parametric PDEs, SIAM J. Sci. Comput. 47 (2) (2025) C501–C528.
[40] F.M. Bersetche, J.P. Borthagaray, A deep first-order system least squares method for solving elliptic PDEs, Comput. Math. Appl. 129 (2023) 136–150.
[41] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs, 2018, http://github.com/jax-ml/jax. v0.3.13.
[42] T. Xue, S. Liao, Z. Gan, C. Park, X. Xie, W.K. Liu, J. Cao, JAX-FEM: a differentiable GPU-accelerated 3D finite element solver for automatic inverse design and mechanistic data science, Comput. Phys. Commun. (2023) 108802.
[43] G. Wu, A framework for structural shape optimization based on automatic differentiation, the adjoint method and accelerated linear algebra, Struct. Multidiscip. Optim. 66 (7) (2023) 151. https://doi.org/10.1007/s00158-023-03601-0
[44] K. Yosida, Functional Analysis, 123, Springer Science & Business Media, 2012.
[45] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, (2014). arXiv preprint arXiv:1412.6980.
[46] R. Frostig, M.J. Johnson, C. Leary, Compiling machine learning programs via high-level tracing, Syst. Mach. Learn. 4 (9) (2018).
[47] D. Aballay, F. Fuentes, V. Iligaray, Á.J. Omella, D. Pardo, M.A. Sánchez, I. Tapia, C. Uriarte, rFEM-NN, 2025, https://github.com/ManuelSanchezUribe/rFEM-NN. GitHub repository.
[48] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, PETSc Web page, 2015, http://www.mcs.anl.gov/petsc.
[49] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41.
[50] Y. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: Neural Networks: Tricks of the Trade, Springer, 2002, pp. 9–50.
[51] I. Babuška, S. Theofanis, The Finite Element Method and its Reliability, Numerical Mathematics and Scientific Computation, New York: Oxford University Press, 2001.
[52] I. Babuška, R.B. Kellogg, J. Pitkäranta, Direct and inverse error estimates for finite elements with mesh refinements, Numer. Math. 33 (4) (1979) 447–471.
[53] J. Schöberl, NETGEN an advancing front 2d/3d-mesh generator based on abstract rules, Comput. Vis. Sci. 1 (1) (1997) 41–52.
[54] J. Schöberl, C+ + 11 implementation of finite elements in NGSolve, Inst. Anal. Sci. Comput. Vienna Univers. Technol. 30 (2014).